目　　录

第 1 章　面向对象编程

本章介绍如何在 MATLAB 中实现面向对象编程。面向对象编程强调使用类和对象来编写程序。使用类和对象，可以向 MATLAB 添加新的数据类型和新的运算符。类描述对象的结构，并定义运算符的类型和可以操作对象的函数。对象是某个类的实例。

1.1　对象和类

可以把类看作是具有特定行为的新的数据类型。例如，Polynomial 类重新定义相加运算符（+）以后，可以对多项式完成正确的相加运算。运算符是类的一种方法。也可以把类看作是可以作为单一实体的新项目。例如 arrow（箭头）对象，它可以显示在图形上，并像句柄图形对象那样具有属性。可以通过简单地实例化 arrow 对象来创建箭头，可以为对象指定用于数据存储的 MATLAB 结构，并创建 M 文件的类目录来将类添加到 MATLAB 环境中。这些 M 文件基于对象进行操作并提供类的方法。类目录中还可以定义应用于对象的不同 MATLAB 运算符，包括算术运算、索引引用等。可以在类中重新定义内部运算符，即进行运算符重载。

1.1.1　面向对象编程的特点

使用设计良好的类时，面向对象编程可以显著提高代码的重用性，并使程序更易于维护和扩展。用类和对象编程与一般的结构化编程有下面一些主要区别。

（1）能实现函数和运算符重载。

（2）可以覆盖已经存在的 MATLAB 函数表示的方法。

（3）把用户定义的对象作为参数调用函数时，MATLAB 首先进行检查，看是否有定义对象类的方法。如果有，MATLAB 调用它。

（4）可以实现数据的封装。

（5）对象属性在命令行中不可见，只能用类方法访问它们。

（6）可以创建类的继承层次。子类可以从一个父类（单继承）和多个父类（多继承）那里进行继承。利用继承，可以共享父函数。

（7）可以用组合创建类，其中一个对象包含另一个对象。这适用于一种对象是另一种对象的一部分的情况。

1.1.2　MATLAB 的数据类层次

在面向对象编程中，所有 MATLAB 数据类都作为类设计成函数。图 1-1 显示了 MATLAB 中定义的 15 种基本数据类型（或类）。可以通过扩展类层次来给 MATLAB 添加一个以结构类继承的用户类。创建的所有类都基于结构。

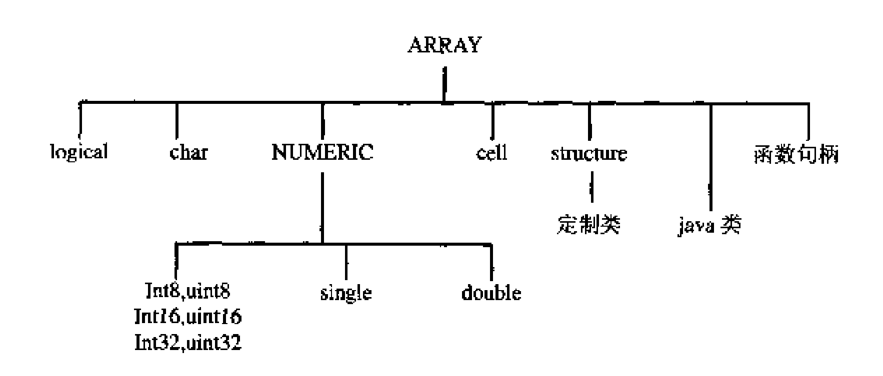


图 1-1 MATLAB 的数据类层次

1.1.3 创建对象

通过调用类的构造函数并传递合适的输入参数，可以创建对象。在 MATLAB 中，构造函数的名称与类名相同，例如，下面的语句

p=polynom([1 0 –2 –5]);

创建一个 polynom 类的对象 p。创建一个 polynom 对象以后，就可以用 polynom 类中定义的方法来操作对象。

1.1.4 调用对象的方法

类方法为 M 函数，该函数将一个对象作为输入参数之一。特定类的方法必须位于该类的类目录中（@classname 目录）。这是 MATLAB 查找类方法的第 1 个位置。

调用对象方法的语法与函数调用类似。通常类似于下面的格式：

[out1,out2,…]=methodName(object,arg1,arg2,…)

例如，自定义类 polynom 具有 char 方法，该方法将 polynom 对象转换为字符串并返回该字符串。下面的语句调用 polynom 对象 p 的 char 方法。

s=char(p);

使用 class 函数，可以确定返回值 s 是否字符串。使用 methods 命令，可以生成为类定义的所有方法的列表。

1.2 在 MATLAB 中创建自己的类

1.2.1 MATLAB 类的方法集合

设计 MATLAB 类时，应该提供一个使类能够在 MATLAB 环境中稳定运行的标准的方法集合。利用这些方法，应该可以实现创建类、设置属性以及对对象的引用、赋值、索引和转换等操作。表 1-1 列出了 MATLAB 类中包括的基本方法。

表 1-1 MATLAB 类中的基本方法

方　法	描　述
类的构造函数	创建一个类对象
display	MATLAB 是否显示对象的内容
set 和 get	设置和获取类属性

续表

方 法	描 述
subsref 和 srbsasgn	用户对象的索引引用和赋值
end	在使用了对象的索引表达式中支持 end 语法，如 A(1:end)
subsindex	支持在索引表达式中使用对象
转换器方法（如 double,char）	将对象转换为一种 MATLAB 数据类型

1.2.2 类目录

1．安装类目录

定义类方法的 M 文件集中在一个目录中，该目录名由类名前加字符“@”组成。例如：本章中用到的一个示例是与单变量多项式有关的一个类。类名和构造函数的名称都是 polynom，定义 polynom 类的 M 文件位于@polynom 目录下。类目录是 MATLAB 搜索路径上目录的子目录。例如，新的@polynom 目录可以是 MATLAB 工作目录的子目录或已经添加到搜索路径中的个人目录。

2．添加类目录到 MATLAB 路径

创建类目录以后，需要更新 MATLAB 路径，这样，MATLAB 才能找到类的源文件。类目录不应该直接把父目录添加到 MATLAB 路径。例如，如果@polynom 类目录的位置是 c:\my_classes\@polynom，用 addpath 命令把类目录添加到 MATLAB 路径中：

```
addpath c:\my_classes;
```

1.2.3 构造函数

构造函数是对类进行操作时第一个运行的函数，它常常用于对象数据的初始化。可以通过调用构造函数并给它传递合适的输入参数来创建类。在 MATLAB 中，构造函数与类具有相同的名称。例如：声明 P=polynom(10 -2 -5)；创建一个属于 polynom 类的名为 P 的对象。一旦创建了 polynom 对象，就可以用类定义的方法操作对象。

特定类的@目录必须包含一个作为类的构造函数的 M 文件，除了@前缀和.m扩展名以外，构造函数的名称与定义类名目录的名称相同。构造函数通过初始化数据结构和实例化类对象来创建对象。

构造函数必须运行一定的函数。这样，对象才能在 MATLAB 环境中正确运行。通常，根据输入参数的个数和类型的不同，构造函数有 3 种，分别对应于没有输入参数、同一个类的一个对象作为输入参数和某种类型的数据作为输入参数的情况。

1．没有输入参数

如果没有输入参数，构造函数将创建一个默认对象。因为没有输入参数，没有数据可用于创建对象，所以用空值或默认值简单地初始化对象的数据，调用类函数实例化对象，并返回对象作为输出参数。在下面两种情况下需要使用本语法：

- 在工作空间中载入对象时，load 函数调用没有参数的构造函数。
- 创建对象数组时，MATLAB 向数组中添加对象。

2. 对象输入参数

如果参数列表中的第一个输入参数是同一类的对象，构造函数将简单地返回对象，使用isa函数确定参数是否为该类对象。

3. 数据输入参数

如果有输入参数，但没有同一类的对象，则构造函数使用输入的数据，常用于使用varargin输入参数有开关控制的程序中。

1.2.4 设置和访问对象数据

需要为类编写方法来提供访问对象数据的途径。set和get方法提供了访问对象数据的便捷途径。

1.2.5 类方法

1. 在MATLAB中设置类方法

类方法实际上是M文件函数，只是该函数的输入参数是对象。类方法必须位于该类的类目录（@class_name目录）中。这是MATLAB查找类方法的第一个位置。调用类方法的语法与调用函数的方法相同，一般地，它与下面的语句行类似：

```
out1 ,out2,…=method_name(object,arg1,arg2,…)
```

例如，名为polynom的类有一个char方法。该方法把polynom对象转换为字符串，并返回该字符串。下面调用polynom对象p的char方法：

```
s=char(p);
```

使用class函数，可以确认返回值s是一个字符串。

```
class(s)
ans =
       char
s
s =
      x^3-2*x-5
```

可以使用methods方法来生成类中的所有方法的列表。

2. 私有方法

私有方法只能被类中的其他方法调用。将有关的M文件放到一个@class_name目录的private子目录中，可以定义私有方法。例如@class_name\private\update_obj.m。update_obj方法的使用范围局限于class_name类。这意味着update_obj可被@class_name目录中定义的方法调用，但它不能被MATLAB命令行或目录路径以外的方法，包括父方法所调用。私有方法和私有函数之间的区别在于，私有方法有一个输入参数是对象，而私有函数却没有。

3. 调试类方法

可以采用与调试其他M文件相同的方法，用MATLAB的调试命令调试对象的方法。惟一的区别在于需要在方法名前添加类的目录，就像下面例子中用dbstop显示的那样。

```
dbstop @polynom/char
```

调试类方法时，已经获得了为类定义的所有方法，包括继承方法、私有方法和私有函数。

1.2.6 引用和赋值

1．索引引用

用户类在 MATLAB 中实现新的数据类型。通过索引引用，可以像访问 MATLAB 内部数据类型一样访问对象数据。例如，A 是 doule 类型的数组，则 A(i)返回 A 的第 i 个元素。

设计类时，可以确定索引引用对于对象的意义。例如，定义一个创建多项式对象的类，各对象包含多项式的系数。

多项式对象的一个索引引用 P(3)能返回 x^3 的系数的值、x=3 时多项式的值或某些不同的值，通过创建两个类方法，即 subsref 和 subsasgn 来定义特定的索引行为。无论什么时候引用索引或指定类对象，MATLAB 都会调用这些方法。如果没有为类定义这些方法，则该类对象的索引就没有定义。

通常，索引对象的规则与索引结构数组的规则相同。

2．下标引用

用对象在赋值语句的右端使用索引或指定字段称为下标引用。在此情况下，MATLAB 调用 subsref 方法。对象下标引用可以有 3 种形式：数组索引、单元数组索引和结构字段名，即

```
A(I)
A{I}
A.field
```

使用这 3 种方法中的任何一种时，MATLAB 都会调用类目录下的 subsref 方法。MATLAB 给 subsref 方法传递两个参数：A 和 S，即

```
B=subsref(A,S)
```

第一个参数 A 是引用的对象，第二个参数 S 是有两个字段的结构数组：S.type 是包含指定索引类型的“()”，“{}”或“.”的字符串。其中，小括号表示数值数组，大括号表示单元数组，点表示结构数组。S.subs 是包含实际下标的单元数组或字符串。用于索引的冒号(:)用字符串“:”传递。例如，表达式 A(1:2,:)会使 MATLAB 调用 subsref(A,S)，其中，S 是一个 1×1 的结构，且

```
S.type='()'
S.subs={1:2,':'}
```

类似地，对于 A{1:2}，有

```
S.type='{}'
S.subs={1:2}
```

对于 A.field，调用 subsref(A,S)，有

```
S.type='.'
S.subs='field'
```

这些简单的调用可以组合成更复杂的索引表达式。此时，length(S)是索引水平数。例如，

```
A(1,2). name(3,4)
```

调用 subsref(A,S)，其中，S 是一个 3×1 的结构数组，其值为：

```
S(1).type='()'        S(1).subs='{1,2}'
S(2).type='.'         S(2).subs='name'
S(3).type='()'        S(3).subs='{3:4}'
```

subsref 方法必须解释 MATLAB 中传递的索引表达式。一个典型用途是使用开关语句确定用到的索引类型，并获取实际的编号。下面 3 段代码演示如何解释输入参数，每种情况下，函数必须返回值 B。

对于数组索引：

```
switch S.type
case '()'
    B = A(S.subs{:});
end
```

对于单元索引：

```
switch S.type
case '{}'
    B = A(S.subs{:}); % A 是一个单元数组
end
```

对于结构索引：

```
switch S.type
case '.'
    switch S.subs
    case 'field1'
        B = A.field1;
    case 'field2'
        B = A.field2;
    end
end
```

3. 索引赋值

用对象在赋值语句的左端使用索引或字段指示符称为索引赋值。此时，MATLAB 调用一个名为 subsasgn 的方法。对象索引赋值可以有 3 种形式：数组索引、单元数组索引和结构字段名。以 A=subsasgn(A,S,B)的形式调用 subsasgn 函数，得到下面的结果：

```
A(I) = B
A{I} = B
A.field = B
```

第一个参数 A 是引用的对象；第二个参数 S 与 subsref 函数具有相同的字段；第三个参数 B 是一个新值。

1.2.7 对象索引

1. 方法中的对象索引

如果类方法中进行了索引引用，MATLAB 使用它的内部 subsref 函数访问方法所属类的数据。如果方法从另一个类访问数据，则 MATLAB 调用该类重载的 subsref 函数。对于索引赋值和 subsasgn，存在同样的情况。

下例显示类 employee 中定义的方法 testref。本方法用 subsref 函数引用对象内的 address 字段。它还在另一个类中引用相同的字段，这时使用该类重载的 subsref 函数。

```
function testref(myclass,otherclass)
  myclass.address                % 使用内部 subsref 函数
  otherclass.address             % 使用重载的 subsref 函数
```

本例创建一个 employee 对象和一个 company 对象。

```
empl = employee('Johnson','Chicago');
comp = company('The MathWorks','Natick');
```

调用 employee 类的方法 testref，MATLAB 只在访问方法所属类以外的数据时才使用重载的 subsref 函数。

```
testref(empl,comp)
ans =                          % 调用了内置的 subsref 函数
    Chicago
ans =                          % 调用了@company\subsref
Executing @company\subsref…
Natick
```

2．end 索引

在对象索引表达式中使用 end 方法时，MATLAB 调用对象的 end 类方法，如果希望在与你的类对象有关的索引表达式中能使用 end 方法，必须给你的类定义一个 end 方法。end 方法的调用格式为：

```
end(a,k,n)
```

其中，a 为用户对象，k 为表达式中 end 语法用到的索引，n 是表达式中索引的总个数。例如，考虑表达式 A(end-1,:)，MATLAB 可以这样使用 end 方法：

```
end(A,1,2)
```

即，共有两个索引元素，end 语句发生在第一个索引元素。然后 end 方法返回第一维最后一个元素的索引值。实现类的 end 方法时，必须保证它返回一个对对象合适的值。

3．用对象索引对象

MATLAB 把对象作为索引时，调用 subsindex 方法。例如，假设有一个对象 a，而且希望使用该对象对另一个对象 b 进行索引，即：

```
C=b(a);
```

subsindex 方法会尽可能简单地将该对象转换为可以用作索引的 double 格式，如下例所示：

```
function d = subsindex(a)
    d = double(a);
```

注意，subsindex 的值基于 0，而不是基于 1。

1.2.8 识别对象

1．isa 函数

用在构造函数中的函数 class 和 isa 也能用在类目录以外，下面表达式

```
isa(a, 'class_name');
```

检查 a 是否指定类的对象。例如，p 是一个 polynom 对象，下面的每个表达式都返回 True：

```
isa(pi,'double');
isa('hello','char');
isa(p,'polynom');
```

在类目录以外，class 函数只有一个参数（只有在构造函数内部，class 函数才能有一个以上的参数），表达式 class(a)返回一个包含 a 的类名的字符串。例如：

```
class(pi),
class('hello'),
class(p)
```

返回

```
'double',
'char',
'polynom'
```

2. whos 函数

用 whos 函数察看 MATLAB 工作空间中的对象。

```
whos
    Name        Size          Bytes         Class
      p         1x1           156       polynom object
```

3. display 方法

对象为不以分号结尾的表达式的结果时，MATLAB 调用一个名为 display 的方法。例如，创建一个 double 型参数 a，调用 MATLAB 的方法为：

```
a = 5
a =
     5
```

应该在类中定义一个 display 方法，这样，从类中引用对象时，MATLAB 能在命令行中显示值。在许多类中，方法能简单地输出参数名，然后使用转换器方法输出内容或参数的值。因为 MATLAB 把输出显示成字符串，所以必须定义 char 方法，将对象的数据转换为字符串。

1.2.9 转换器方法

转换器方法是一个类方法，它与另一个类具有相同的名称，如 char 或 double 等。转换器方法把某类的一个对象作为输入，返回其他类的对象，调用格式为：

```
b=class_name(a)
```

其中，a 是一个不同于 class_name 的类对象。

1.3 重载

在许多情况下，当参数是对象时，可能希望改变 MATLAB 运算符和函数的行为。通过重载有关的函数可以达到此目的。重载使一个函数可以有不同的形式、不同数目的输入参数，并能适应对象的不同情况进行操作。

1.3.1 运算符重载

每个 MATLAB 内部运算符都有一个相关的函数名，如“+”运算符的相关函数名为 plus.m。可以在类目录中通过创建具有对应名称的 M 文件来重载任何运算符。例如，如果 p 或 q 是 class_name 类型的对象，表达式 p+q 生成一个对函数@class_name/plus.m 的调用。如果 p 和 q 是不同类的对象，则 MATLAB 采用优先原则确定先用哪个方法（关于优先原则，请参见 1.7 节的内容）。

1.3.2 函数重载

可以通过创建与类目录中名称相同的函数来重载任何函数。函数调用对象时，总是首先在搜索路径中的类目录中进行查找，然后才是在其他位置中查找。例如，要重载 plot 函数，只需要简单地把你自己的 plot.m 放到类目录中就行了。

1.3.3 示例——polynom 类

下面通过定义一个名为 polynom 的新类，实现描述多项式的 MATLAB 数据类型。该类为保存数据指定一个结构，并定义操作 polynom 对象的方法目录(@polynom)。

1. polynom 类的数据结构

polynom 类用行矢量表示多项式，行矢量中包含降序排列的参数幂次项的系数。这样，polynom 对象 p 是一个具有单一字段 p.c 的结构，其中包含各系数。只有@polynom 目录中的方法才可以访问本字段。

2. polynom 类的方法

为了创建一个在 MATLAB 环境中运行良好的类，并为多项式数据类型提供有用的功能，polynom 类实现下面的方法：

- 构造函数 polynom.m
- polynom 类型向 double 类型转换的函数
- polynom 类型向 char 类型转换的函数
- display 方法
- subsref 方法
- 重载的 +, - 和 * 运算符
- 重载的 roots, polyval, plot 和 diff 函数

（1）polynom 类的构造函数

下面是 polynom 类的构造函数@polynom/polynom.m。

```
function p = polynom(a)
if nargin == 0
    p.c = [];
    p = class(p,'polynom');
elseif isa(a,'polynom')
    p = a;
else
    p.c = a(:).';
    p = class(p,'polynom');
end
```

polynom 类有 3 个构造函数，它们具有不同的形式。

- 没有输入参数：如果调用没有参数的构造函数，它返回一个带空字段的 polynom 对象。
- 输入参数为对象：如果调用输入参数为对象的构造函数，MATLAB 返回该对象。
- 输入参数为系数矢量：如果输入参数是一个变量，该变量不是 polynom 对象，则将它重塑为行矢量，并将它赋为对象结构的 c 字段，class 函数创建 polynom 对象，并在后面由构造函数返回。下面是一个使用 polynom 类构造函数的例子：

```
p=polynom(10 -2 -5)
```

这将创建一个指定系数的多项式。

（2）转换器方法

转换器方法将一个类的对象转换为另一个类的对象。包含在 MATLAB 类中的最重要的转换器方法中的两个，即 double 和 char。转换为 double 型将生成 MATLAB 传统矩阵，尽管它对某些类可能不合适。转换为 char 型对于生成打印输出很有用。

● polynom 类型到 double 类型的转换：polynom 类型到 double 类型的转换方法是一个简单的 M 文件：@polynom double.m。它只提取对象 p 的系数矢量 p=polynom(1 0 -2 -5)。

```
function c = double(p)
c = p.c;
```

语句 double(p)返回

```
ans=
    1    0    -2    -5
```

● polynom 类型到 char 类型的转换：polyuom 类型到 char 类型的转换方法是一个关键的方法，因为它生成与独立变量 x 的幂次有关的字符串。所以，一旦已经指定了 x，返回的字符串就是一个语法上正确的 MATLAB 表达式。这里为@polynom/char.m。

```
function s = char(p)
    if all(p.c == 0)
        s = '0';
    else
        d = length(p.c) - 1;
        s = [];
    for a = p.c;
        if a ~= 0;
            if ~isempty(s)
                if a > 0
                    s = [s ' + '];
                else
                    s = [s ' - '];
                    a = -a;
                end
            end
            if a ~= 1 | d == 0
                s = [s num2str(a)];
                if d > 0
                    s = [s '*'];
                end
            end
            if d >= 2
                s = [s 'x^' int2str(d)];
            elseif d == 1
                s = [s 'x'];
            end
        end
        d = d - 1;
    end
end
```

如果创建 polynom 对象 p(p=polynom(1 0 –2 –5);)，然后对 p 调用 char 方法：

```
char(p)
```

则 MATLAB 生成如下结果：

```
ans =
    x^3 - 2*x - 5
```

char 返回的值是一个字符串，一旦为 x 定义了标量值，可以将它传递给 eval 函数。例如：

```
x = 3;
eval(char(p))
ans =
    16
```

（3）polynom 类的 display 方法

其 M 文件为@polynom/display.m。本方法依赖于 char 方法，生成一个表示多项式的字符串，然后显示在屏幕上。本法生成的输出与 MATLAB 的标准输出相同。即参数名后面跟等号，然后空一行，再在一个新行中显示值。

```
function display(p)
disp(' ');
disp([inputname(1),' = '])
disp(' ');
disp(['     ' char(p)])
disp(' ');
```

语句 p=polynom(1 0 –2 –5) 创建一个 polynom 对象，因为语句不以分号结束，结果输出为：

```
p =
    x^3 - 2*x - 5
```

（4）polynom 的 subsref 方法

对于 polynom 对象 p

```
p = polynom([1 0 -2 -5]);
```

下面的下标表达式返回 x=3 和 x=4 处多项式的值：p([3 4])

```
ans =
    16    51
```

subsref 方法利用 polynom 类中定义的 char 方法，生成一个可以计算的表达式。

```
function b = subsref(a,s)
    switch s.type
    case '()'
      ind = s.subs{:};
      for k = 1:length(ind)
          b(k) = eval(strrep(char(a),'x',num2str(ind(k))));
      end
    otherwise
      error('Specify value for x as p(x)')
end
```

用 char 方法生成多项式表达式以后，使用 strrep 函数交换字符 x，以传递值的内容。然后 eval 函数评价表达式并在输出参数中返回值。

(5) 重载算术运算符

有些算术运算符对于多项式是有意义的，应该在 polynom 类中实现。重载算术运算符时，记住进行操作的数据类型。本例中，为 polynom 类定义了 plus, minus 和 mtimes 方法，以便实现加、减法和乘法等运算。

● 重载"+"运算符：如果 p 或 q 为 polynom 类对象，表达式 p+q 调用@polynom/plus.m 函数，下面的函数重新为 polynom 类定义"+"运算符。

```
function r = plus(p,q)
  p = polynom(p);
  q = polynom(q);
  k = length(q.c) - length(p.c);
  r = polynom([zeros(1,k) p.c] + [zeros(1, -k) q.c]);
```

该函数首先将两个输入参数定义为多项式。这样保证至少有一个输入不是多项式的情况下函数也能正常运行。然后函数获取两个系数矢量，并且在必要时将其中的某个置 0，以使它们具有相同的长度。实际求和是求两个多项式的系数的和，用矢量表示。最后，函数第三次调用 polynom 构造函数，得到结果。

● 重载"-"运算符：可以用相同的方法实现"-"运算符。MATLAB 调用@polynom/minus.m 计算 p-q。

```
function r = minus(p,q)
  p = polynom(p);
  q = polynom(q);
  k = length(q.c) - length(p.c);
  r = polynom([zeros(1,k) p.c] - [zeros(1,-k) q.c]);
```

● 重载"*"运算符：MATLAB 调用方法@polynom/mtimes.m 计算 p*q。因为它从 MATLAB 矩阵乘法中继承而来，所以函数名的第一个字母为 m。

```
function r = mtimes(p,q)
  p = polynom(p);
  q = polynom(q);
  r = polynom(conv(p.c,q.c));
```

● 使用重载的运算符：给定 polynom 对象 p=polynom([1 0 -2 -5])，MATLAB 在使用语句 q=p+1 时调用函数@polynom/plus.m 和@polynom/mtimes.m。

```
q = p+1
r = p*q
```

生成

```
q =
    x^3 - 2*x - 4
r =
    x^6 - 4*x^4 - 9*x^3 + 4*x^2 + 18*x + 20
```

(6) 重载函数

MATLAB 已经有几个可以操作多项式的函数，它们可以通过重载，用在新的 polynom 对

象中。在许多情况下，重载方法可以将原始函数应用于系数字段。

- 为 polynom 类重载 roots 方法：方法@polynom/roots.m计算 polynom 对象所代表的多项式函数的根。

```
function r = roots(p)
    r = roots(p.c);
    The statement roots(p)
    results in ans =
        2.0946
        -1.0473 + 1.1359i
        -1.0473 - 1.1359i
```

- 为 polynom 类重载 polyval 方法：polyval 方法计算给定点集的多项式。@polynom/polyval.m 使用嵌套的乘法或 Horner 法减少计算 x 的不同幂次的乘法操作次数。

```
function y = polyval(p,x)
y = 0;
for a = p.c
    y = y.*x + a;
end
```

- 为 polynom 类重载 plot 方法：用 root 和 polyval 重载 plot 方法。

```
function plot(p)
    r = max(abs(roots(p)));
    x = (-1.1:0.01:1.1)*r;
    y = polyval(p,x);
    plot(x,y);
    title(char(p))
    grid on
```

- 重载 diff 方法：方法@polynom/diff.m 对一个多项式进行求导。方法是幂次减 1，用原来的幂次乘上对应的系数。

```
function q = diff(p)
c = p.c;
d = length(c) - 1;   % degree
q = polynom(p.c(1:d).*(d:  -1:1));
```

下面用 methods 命令列出类的所有方法。

```
methods polynom
Methods for class polynom:
chardisplayminusplotpolynomrootsdiffdoublemtimespluspolyvalsubsref
```

下面先对两个 polynom 对象 x 和 p 作运算，然后绘图。

```
x = polynom([1 0]);
p = polynom([1 0 -2 -5]);
plot(diff(p*p + 10*p + 20*x) - 20)
```

生成图 1-2。

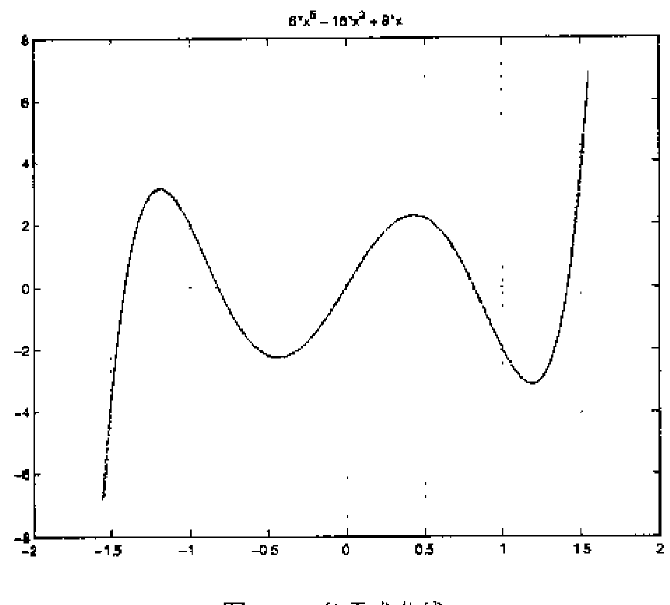


图 1-2　多项式曲线

1.4　继承

MATLAB 对象可以从另一个 MATLAB 对象那里继承属性和行为。继承时，子对象包含父对象的所有字段，并且可以调用父对象的方法。父方法可以获取子对象从父类那里继承的字段，但不能获取子对象的新字段。

继承是面向对象编程的关键特征。它通过允许子对象利用已经存在的父对象的代码，使得代码重用更简单。继承使子对象的行为与父对象的完全类似，为具有相似行为但实现方式不同的相关类的开发提供了方便。

有两种继承类型：

- 简单继承：子对象从一个父类那里继承特性。
- 多继承：子对象从多个父类那里继承特性。

1.4.1　简单继承

类只从一个父类那里继承属性并给自己添加新属性时，使用简单继承。继承意味着属于子类的对象具有与父类相同的字段，以及其他字段。所以，父类的方法可以操作属于子类的对象。但是子类的方法不能操作属于父类的对象。不能直接从子类获取父类的字段，必须使用为父类定义的访问方法来进行访问。

继承另一个类的行为的类的构造函数具有两个特点：

- 调用父类的构造函数来创建继承的字段。
- class 函数的调用语法稍微有些差别，既反映子类的特点又反映父类的特点。

用 class 函数创建简单继承关系的一般语法格式为：

```
childObj = class(childObj, 'childClass', parentObj);
```

简单继承可以扩展到多层。如果父类本身是从其他类继承来的，则其子类会自动继承它祖父类的特性。这通常称为多层继承。

父类不清楚子类的属性或方法。子类不能直接获取父属性，而必须使用父类的访问方法

（例如 get 或 subsref 方法）来访问父属性。利用子类的方法，通过子类结构中的父字段来实现该访问。例如，下面用构造函数创建了对象 c 时，

```
c = class(c,'childClassname', parentObject);
```

MATLAB 自动在对象的包含父对象的结构中创建一个字段 c.parentClassname。然后可以在子对象调用父对象 display 方法的 display 方法中添加下面一行语句：

```
display(c.parentClassname)
```

1.4.2 多继承

多继承时，对象可以从多个父类那里继承属性。子对象具有所有父类的字段和它本身的字段。多继承也可以包含多个继承层次，例如，每个父对象可以具有从多个祖父对象那里继承来的字段。通过调用具有 3 个以上参数的 class 函数，可以用构造函数实现多继承。

```
obj=class(structure,'Classname',parent1,parent2,...)
```

可以根据需要在 class 函数的输入列表中添加多个父参数。

多个父类可以具有名称相同的方法，此时，MATLAB 调用构造函数中第 1 个出现的父类的该方法。不能访问后面具有该名称的父函数。

1.4.3 示例——asset 类及其子类

作为简单继承的示例，考虑一个一般的财产类，它用于表示所有值钱的东西，比如股票、债券、存款等。设计这个类集时，用 asset 类包含所有特殊财产子类的公共数据。单个的财产子类，例如 stock 类继承 asset 类的属性，并具有其他属性。子类表示一种财产。

1．继承模型

图 1-3 显示了 asset 类及其子类的继承关系。stock、bond 和 savings 类从 asset 类那里继承结构字段。本例中，asset 类保存所有子类的公共属性，并提供共有的方法。本例显示了如何实现 asset 类和 stock 类。bond 类和 savings 类的实现方式与 stock 类的相似。

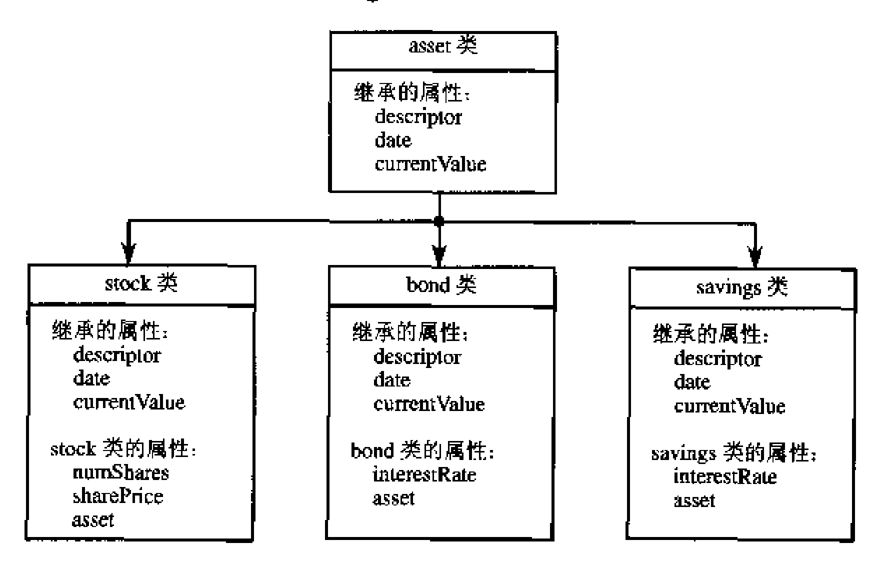


图 1-3　asset 类及其子类的继承关系

2．asset 类的设计

asset 类提供对其所有子类公共信息的保存和访问能力。因为不准备直接创建该类的实例，

所以不需要扩展方法集。为了实现数据保存和访问，该类需要包含下面几个方法：

- 构造函数
- get 和 set 函数
- subsref 和 subsasgn 函数
- display 函数

3. asset 类的其他方法

asset 类为其子类提供数据保存，但不直接进行实例化。利用 set、get 和 display 方法可以访问所保存的数据。不必实现 asset 对象的所有方法（例如转换器方法、end 方法和 subsindex 方法等），因为只有子类访问数据。

4. asset 类的构造函数

asset 类基于有 4 个字段的结构数组创建：

- descriptor：某类财产的标识符（例如股票名称、存款账号等）。
- date：创建对象的日期（用 date 命令计算）。
- type：财产的类型（如存款、债券、股票等）。
- currentValue：财产的当前数目（根据子类数据计算）。

这些信息对于 asset 类的子对象来说是公共的，所以从父对象中进行控制，以避免在每个子类中定义相同的字段。当子类个数增加时，这非常有用。

```
function a = asset(varargin)
% asset 对象的构造函数
% a = asset(descriptor, currentValue)
switch nargin
case 0
% 如果没有输入参数，创建一个默认对象
    a.descriptor = 'none';
    a.date = date;
    a.type = 'none';
    a.currentValue = 0;
    a = class(a,'asset');
case 1
% 如果只有一个参数，返回它
    if (isa(varargin{1},'asset'))
        a = varargin{1};
    else
        error('Wrong argument type')
    end
case 3
% 用指定值创建对象
    a.descriptor = varargin{1};
    a.date = date;
    a.type = varargin{2};
    a.currentValue = varargin{3};
    a = class(a,'asset');
otherwise
    error('Wrong number of input arguments')
end
```

函数使用 switch 语句将 3 种可能情况组合到一起：

- 没有参数时，构造函数返回一个默认的 asset 对象。
- 有一个参数时，对象简单地返回它。
- 有两个参数时，构造函数返回一个新的 asset 对象。

不直接调用 asset 类的构造函数，而是从子类构造函数中调用它，因为提供它们的目的是保存公共数据。

5. asset 类的 get 方法

asset 类需要提供访问包含在 asset 对象中数据的方法。下面的函数实现该方法。它使用大写属性名提供与其他 MATLAB 对象类似的接口。

```
function val = get(a, propName)
% 从指定对象那里获取 asset 类的属性
% 并返回它
switch propName
case 'Descriptor'
    val = a.descriptor;
case 'Date'
    val = a.date;
case 'CurrentValue'
    val = a.currentValue;
otherwise
    error([propName,' Is not a valid asset property'])
end
```

该函数接受一个对象和属性名输入，并使用 switch 语句确定访问哪个字段。在继承的属性中访问数据时，该方法被子类的 get 方法调用。

6. asset 类的 set 方法

asset 类的 set 方法被子类的 set 方法调用。该方法接受一个 asset 对象和一个可变长度的属性名/值对参数列表作为输入，返回修改后的对象。

```
function a = set(a,varargin)
% 设置 asset 类的属性并返回更新后的对象
propertyArgIn = varargin;
while length(propertyArgIn) >= 2,
    prop = propertyArgIn{1};
    val = propertyArgIn{2};
    propertyArgIn = propertyArgIn(3:end);
    switch prop
    case 'Descriptor'
        a.descriptor = val;
    case 'Date'
        a.date = val;
    case 'CurrentValue'
        a.currentValue = val;
    otherwise
        error('Asset properties: Descriptor, Date, CurrentValue')
    end
end
```

子类的 set 方法调用 asset 类的 set 方法，并需要有返回修改后对象的能力，因为 MATLAB 不支持以引用方式传递参数。

7. asset 类的 subsref 方法

使用 subsref 方法访问包含在 asset 对象中的数据，访问时使用以 1 为底的数值索引和结构字段名称索引。外部的 switch 语句确定索引类型是数值索引还是字段名称索引。内部的 switch 语句将索引映射到合适的值。

不管什么时候以下标形式引用对象(例如 A(i)或 A.fieldname)，MATLAB 都会调用 subsref 函数。

```
function b = subsref(a,index)
% 为 asset 对象定义字段名索引
switch index.type
case '()'
    switch index.subs{:}
    case 1
        b = a.descriptor;
    case 2
        b = a.date;
    case 3
        b = a.currentValue;
    otherwise
        error('Index out of range')
    end
case '.'
    switch index.subs
    case 'descriptor'
        b = a.descriptor;
    case 'date'
        b = a.date;
    case 'currentValue'
        b = a.currentValue;
    otherwise
        error('Invalid field name')
    end
case '{}'
    error('Cell array indexing not supported by asset objects')
end
```

8. asset 类的 subsasgn 方法

subsasgn 方法与 subsref 方法类似，但用于赋值。使用它可以用以 1 为底的数值索引和结构字段名称索引改变包含在对象中的数据。外部的 switch 语句确定索引类型是数值索引还是字段名称索引。内部的 switch 语句将索引映射到合适的值。

不管什么时候执行赋值语句(例如 A(i)=val,A{i}=val 或 A.fieldname=val)，MATLAB 都会调用 subsasgn 函数。

```
function a = subsasgn(a,index,val)
% 为 asset 对象定义索引
switch index.type
```

```
case '()'
    switch index.subs{:}
    case 1
        a.descriptor = val;
    case 2
        a.date = val;
    case 3
        a.currentValue = val;
    otherwise
        error('Index out of range')
    end
case '.'
    switch index.subs
    case 'descriptor'
        a.descriptor = val;
    case 'date'
        a.date = val;
    case 'currentValue'
        a.currentValue = val;
    otherwise
        error('Invalid field name')
    end
end
```

使用 subsasgn 方法可以用两种方法给 asset 对象数据结构赋值。例如，假设有一个子 stock 对象 s

```
s = stock('XYZ',100,25);
```

在 stock 类的方法中，可以用下面两条语句中的一条改变 descriptor 字段的值。

```
s.asset(1) = 'ABC';
```

或

```
s.asset.descriptor = 'ABC';
```

9. asset 类的 display 方法

设计 asset 类的 display 方法，是为了从子类的 display 方法中调用它。其目的是显示它为子对象保存的数据。该方法简单地格式化要显示的数据，显示格式与子对象 display 方法的保持一致。

```
function display(a)
% 显示一个 asset 对象
stg = sprintf(...
    'Descriptor: %s\nDate: %s\nType: %s\nCurrent Value:%9.2f',...
    a.descriptor,a.date,a.type,a.currentValue);
disp(stg)
```

现在，stock 类的 display 方法可以调用该方法来显示保存在父类中的数据。

10. asset 类的 fieldcount 方法

asset 类的 fieldcount 方法返回 asset 对象数据结构中的字段数。使用 fieldcount 方法，使 asset 类的子类方法可以在执行过程中确定 asset 对象中的字段个数，而不需要子类方法了解

asset 类。这使得可以在不改变子类方法的条件下改变 asset 类数据结构中的字段数。

```
function numFields = fieldcount(assetObj)
% 确定 asset 对象中的字段数
% 用于 asset 类子类的方法
numFields = length(fieldnames(assetObj));
```

struct 函数将对象转换为它的等价数据结构，使得可以访问结构的内容。

11．设计 stock 类

用 stock 类表示股票投资。该类包括 2 个自己的属性和 3 个从父对象继承的属性。stock 类自己的属性为 numShares 和 sharePrice，分别表示股票数和每份股票的价格；从 asset 类继承的属性为 descriptor、date 和 currentValue，descriptor 为该种财产的标识符，date 为创建对象的日期，currentValue 为财产的当前值。

注意，实际上属性名与 stock 和 asset 对象内部使用的结构数组的字段名不相同。属性名接口由 stock 和 asset 的 set 和 get 方法控制，其设计类似其他 MATLAB 对象属性的接口。

stock 对象结构中的 asset 类字段包含父 asset 对象，用于访问父结构中的继承字段。

12．stock 类方法

stock 类实现了下面一些方法：

- 构造函数
- get 和 set 方法
- subsref 和 subsasgn 方法
- display 方法

13．stock 类的构造函数

stock 类的构造函数用 3 个输入参数创建 stock 对象：

- 股票名称
- 持股数
- 股票价格

构造函数必须从 stock 类构造函数内部创建一个 asset 对象，作为 stock 对象的父对象。所以，stock 构造函数必须调用 asset 类的构造函数。被调用来创建 stock 对象的 class 函数将 asset 对象作为父对象定义。

记住，asset 对象是在 stock 类构造函数的暂时工作空间中创建的，在 stock 类结构中作为字段(.asset)保存。stock 对象继承 asset 类的字段，但 asset 对象不会返回到基本工作空间。

```
function s = stock(varargin)
% stock 类的构造函数
% s = stock(descriptor, numShares, sharePrice)
switch nargin
case 0
% 如果没有输入参数，创建一个默认对象
    s.numShares = 0;
    s.sharePrice = 0;
    a = asset('none',0);
    s = class(s, 'stock',a);
case 1
```

```
% 如果有一个参数，返回它
    if (isa(varargin{1},'stock'))
        s = varargin{1};
    else
        error('Input argument is not a stock object')
    end
case 3
% 用指定值创建对象
    s.numShares = varargin{2};
    s.sharePrice = varargin{3};
    a = asset(varargin{1},'stock',varargin{2} * varargin{3});
    s = class(s,'stock',a);
otherwise
    error('Wrong number of input arguments')
end
```

14. 构造函数调用语法

stock 类的构造函数可以用下面 3 种方式中的一种调用：

- 没有输入参数：如果调用时没有输入参数，则构造函数返回默认对象，字段为空。
- 输入参数是一个 stock 对象：如果输入参数为惟一的 stock 对象，则构造函数返回该输入参数。如果输入参数不是 stock 对象，会出错。
- 3 个输入参数：如果有 3 个输入参数，构造函数用它们定义 stock 对象。

如果不满足上面 3 个条件，返回一则错误。例如，下面的语句创建一个 stock 对象来记录 100 股 XYZ 公司的股票，股票价格为每股 25 美元。

```
XYZStock = stock('XYZ',100,25);
```

15. stock 类的 get 方法

与句柄图形类似，get 方法提供了用"propertyname"类型的接口访问 stock 对象数据的一种途径。本例中，属性名与结构字段名类似，可以有很大的不同。也可以选择用 get 方法剔除某些字段，或者，如果该行为与设计吻合，可以从同一字段给多个属性名返回数据。

```
function val = get(s,propName)
% 从指定对象获取 stock 对象的属性
% 并返回值。属性名为：NumberShares
% SharePrice, Descriptor, Date, CurrentValue
switch propName
case 'NumberShares'
    val = s.numShares;
case 'SharePrice'
    val = s.sharePrice;
case 'Descriptor'
    val = get(s.asset,'Descriptor'); % 调用 asset 对象的 get 方法
case 'Date'
    val = get(s.asset,'Date');
case 'CurrentValue'
    val = get(s.asset,'CurrentValue');
otherwise
    error([propName ,'Is not a valid stock property'])
end
```

注意，asset 对象通过 stock 对象的 asset 字段(s.asset)进行访问。调用带父参数的 class 函数时，MATLAB 自动创建该字段。

16. stock 类的 set 方法

set 方法提供了与 get 方法类似的"propertyname"接口。用它更新股票数、股票价格和股票名称。当前值和日期自动进行更新。

```
function s = set(s,varargin)
% 将 stock 属性设置为指定值
% 并返回更新后的对象
propertyArgIn = varargin;
while length(propertyArgIn) >= 2,
    prop = propertyArgIn{1};
    val = propertyArgIn{2};
    propertyArgIn = propertyArgIn(3:end);
    switch prop
    case 'NumberShares'
        s.numShares = val;
    case 'SharePrice'
        s.sharePrice = val;
    case 'Descriptor'
        s.asset = set(s.asset,'Descriptor',val);
    otherwise
        error('Invalid property')
    end
end
s.asset = set(s.asset,'CurrentValue',...
            s.numShares * s.sharePrice,'Date',date);
```

注意：本函数用新值创建和返回一个新的 stock 对象，然后覆盖旧值。例如，对于 stock 对象

```
s = stock('XYZ',100,25);
```

下面的 set 命令更新股票价格：

```
s = set(s,'SharePrice',36);
```

覆盖原来的 stock 对象是必要的，因为 MATLAB 不支持以引用方式传递参数。所以 set 命令实际上是在操作对象的复制。

17. stock 对象的 subsref 方法

subsref 方法为 stock 类定义下标索引。本例中，subsref 函数实现了 stock 对象的数值索引和结构字段名称索引。

```
function b = subsref(s,index)
% 为 stock 对象定义字段名称索引
fc = fieldcount(s.asset);
switch index.type
case '()'
    if (index.subs{:} <= fc)
        b = subsref(s.asset,index);
    else
        switch index.subs{:} - fc
```

```
            case 1
                b = s.numShares;
            case 2
                b = s.sharePrice;
            otherwise
            error(['Index must be in the range 1 to ',num2str(fc + 2)])
            end
        end
    case '.'
        switch index.subs
        case 'numShares'
            b = s.numShares;
        case 'sharePrice'
            b = s.sharePrice;
        otherwise
            b = subsref(s.asset,index);
        end
    end
```

外面的 switch 语句确定索引是数值索引还是字段名索引。asset 类的 fieldcount 方法确定 asset 结构中的字段数，if 语句为索引 1 到 fieldcount 调用 asset 的 subsref 方法。

大于 fieldcount 返回个数的数值索引由里面的 switch 语句控制，它将索引值映射为 stock 结构中的合适字段。字段名索引假设 numShares 和 sharePrice 以外的字段名为 asset 字段。asset 类的 subsref 方法进行字段名错误检查。

18. stock 类的 subsasgn 方法

使用 subsasgn 方法，用数值索引和结构字段名称索引改变 stock 对象中包含的数据。不管什么时候执行赋值语句，MATLAB 都会调用 subsasgn 方法。

```
    function s = subsasgn(s,index,val)
    % 为 stock 对象定义索引赋值
    fc = fieldcount(s.asset);
    switch index.type
    case '()'
        if (index.subs{:} <= fc)
            s.asset = subsasgn(s.asset,index,val);
        else
            switch index.subs{:}-fc
            case 1
                s.numShares = val;
            case 2
                s.sharePrice = val;
            otherwise
            error(['Index must be in the range 1 to ',num2str(fc + 2)])
            end
        end
    case '.'
        switch index.subs
        case 'numShares'
            s.numShares = val;
```

```
            case 'sharePrice'
                s.sharePrice = val;
            otherwise
                s.asset = subsasgn(s.asset,index,val);
            end
        end
```

外部的 switch 语句确定索引是数值索引还是字段名索引。asset 类的 fieldcount 方法确定 asset 结构中的字段数。if 语句中，索引 1 到 fieldcount 调用 asset 的 subsasgn 方法。

大于 fieldcount 返回个数的数值索引由里面的 switch 语句控制，它将索引值映射为 stock 结构中的合适字段。字段名索引假设 numShares 和 sharePrice 以外的字段名为 asset 字段。asset 类的 subsasgn 方法进行字段名错误检查。

使用 subsasgn 方法使得可以用两种技巧给 stock 对象数据结构指定值。例如，假设有一个 stock 对象

```
s = stock('XYZ',100,25)
```

可以用下面两种语句中的任何一种改变 descriptor 字段。

```
s(1) = 'ABC';
```

或

```
s.descriptor = 'ABC';
```

19. stock 类的 display 方法

在命令窗口键入下面的命令行

```
XYZStock = stock('XYZ',100,25)
```

MATLAB 在@stock 目录中查找 display 方法。stock 类的 display 方法生成下面的输出结果：

```
Date: 17-Nov-1998
Type: stock
Current Value: 2500.00
Number of shares: 100
Share price: 25.00
Descriptor: XYZ
```

下面是 stock 类的 display 方法。

```
function display(s)
% 显示 stock 对象
display(s.asset)
stg = sprintf('Number of shares: %g\nShare price: %3.2f\n',...
    s.numShares,s.sharePrice);
disp(stg)
```

首先，父 asset 对象传递给 asset 类的 display 方法，显示它的字段。stock 对象的字段用格式化文本字符串以类似的方式进行显示。

注意，如果不实现 stock 类的 display 方法，MATLAB 会调用 asset 类的 display 方法。此时运行仍然正常，但会只显示股票名称、日期、类型和当前值。

1.5 组合

除了继承外，MATLAB 还支持组合，即一个对象可以将另一个对象作为它的字段进行包含。例如，一个有理模型对象可能会使用两个多项式对象，一个作为分子，一个作为分母。

只能用外部对象的方法调用所包含的对象的方法。确定调用函数的哪个版本时，MATLAB 只将对象的最外层包含类作为参数传递，任何被包含对象所属的类被忽略。

组合将一个类包含到另一个类中。基本关系是每个被包含的类是容器类的一部分。例如，考虑将一个资产类作为一系列财产（股票、债券、存款等）的容器类。将各种单独的财产组合到一起以后，可以对它们进行分析，并返回有用的信息。所包含的对象不能直接访问，只能通过资产类的方法进行访问。

1．设计 portfolio 类

用 portfolio 类描述资产，它包含某人的各种财产并提供与其资产状况有关的信息。本例实现一个比较简单的 portfolio 类，该类有下面几个特点和功能。

- 包含个人财产。
- 显示与资产内容有关的信息。
- 显示一个三维饼图，表示总资产中不同类别的财产所占的份额。

2．portfolio 类的方法

portfolio 类只实现 3 个方法，即

- portfolio 方法：为构造函数。
- display 方法：显示与资产内容有关的信息。
- pie3 方法：为 MATLAB pie3 函数的重载版本，参数只有一个 portfolio 对象。

因为 portfolio 对象包含其他对象，portfolio 类的方法可以使用所包含的对象的方法。例如，portfolio 对象的 display 方法可以调用 stock 类的 display 方法，等等。

3．portfolio 类的构造函数

portfolio 类的构造函数将一个客户名称和一个可变长度的子类对象财产列表作为输入参数。portfolio 对象使用具有下列字段的结构数组。

- name：客户名称。
- indAssets：子类对象数组。
- totalValue：所有财产的总额。构造函数计算作为参数传入的对象的财产总额。
- accountNumber：账号。该字段只有在保存 portfolio 对象时进行赋值。

portfolio 类的构造函数如下所示：

```
function p = portfolio(name,varargin)
% 创建一个包含客户名称和财产列表
% 的 portfolio 对象
switch nargin
case 0
    % 如果没有输入参数，创建默认对象
    p.name = 'none';
    p.totalValue = 0;
```

```
        p.indAssets = {};
        p.accountNumber = '';
        p = class(p,'portfolio');
    case 1
        % 如果有一个 portfolio 类的输入参数，返回它
        if isa(name,'portfolio')
            p = name;
        else
            disp([inputname(1) ' is not a portfolio object'])
            return
        end
    otherwise
        % 用指定参数创建对象
        p.name = name;
        p.totalValue = 0;
        for k = 1:length(varargin)
            p.indAssets(k) = {varargin{k}};
            assetValue = get(p.indAssets{k},'CurrentValue');
            p.totalValue = p.totalValue + assetValue;
        end
        p.accountNumber = '';
        p = class(p,'portfolio');
    end
```

4. 构造函数的调用语法

portfolio 类的构造函数方法可以用下面 3 种方法中的一种进行调用：

- 没有输入参数：如果调用时没有输入参数，返回一个具有空字段的对象。
- 输入参数是一个对象：如果输入参数是一个 portfolio 对象，MATLAB 返回输入参数。可以用 isa 函数进行检查。
- 参数多于 2 个：如果参数多于 2 个，构造函数认为第 1 个参数是客户名称，其余参数为财产子类。一个更保险的方法是对输入参数进行检查。例如，使用 isa 函数确定参数是否是正确的对象类。

5. portfolio 类的 display 方法

portfolio 类的 display 方法通过调用对象的 display 方法，列出所包含的每个对象的内容。然后列出客户名和资产总额。

```
function display(p)
% 显示 portfolio 对象
for k = 1:length(p.indAssets)
    display(p.indAssets{k})
end
stg = sprintf('\nAssets for Client:  %s\nTotal Value:  %9.2f\n',...
p.name,p.totalValue);
disp(stg)
```

6. portfolio 类的 pie3 方法

portfolio 类重载 MATLAB 的 pie3 函数，输入参数为一个 portfolio 对象，显示一个三维饼图，表示各种财产之间的比例关系。不管什么时候，只要输入参数是一个 portfolio 对象，

MATLAB 都会调用 pie3 的@portfolio\pie3.m 版本。

```
function pie3(p)
% 创建资产数据的 3 维饼图
stockAmt = 0; bondAmt = 0; savingsAmt = 0;
for k = 1:length(p.indAssets)
    if isa(p.indAssets{k}, 'stock')
        stockAmt = stockAmt + ...
            get(p.indAssets{k}, 'CurrentValue');
    elseif isa(p.indAssets{k}, 'bond')
        bondAmt = bondAmt + ...
            get(p.indAssets{k}, 'CurrentValue');
    elseif isa(p.indAssets{k}, 'savings')
        savingsAmt = savingsAmt + ...
            get(p.indAssets{k}, 'CurrentValue');
    end
end
i = 1;
if stockAmt ~= 0
    label(i) = {'Stocks'};
    pieVector(i) = stockAmt;
    i = i + 1;
end
if bondAmt ~= 0
    label(i) = {'Bonds'};
    pieVector(i) = bondAmt;
    i = i + 1;
end
if savingsAmt ~= 0
    label(i) = {'Savings'};
    pieVector(i) = savingsAmt;
end
pie3(pieVector, label)
set(gcf, 'Renderer', 'zbuffer')
set(findobj(gca, 'Type', 'Text'), 'FontSize', 14)
cm = gray(64);
colormap(cm(48:end, :))
stg(1) = {['Portfolio Composition for ', p.name]};
stg(2) = {['Total Value of Assets: $', num2str(p.totalValue)]};
title(stg, 'FontSize', 12)
```

上面重载的 pie3 方法主要有以下 3 方面的内容：

- 首先用财产子类的 get 方法访问所包含的每个子类的 CurrentValue 属性。将每个类的财产总额加起来。
- 其次，根据所存在的对象，创建饼图标签并生成图形数据组成的矢量。
- 第 3 部分，调用 MATLAB 的 pie3 函数，调整某些字体和颜色查找表，并添加标题。

7. 使用 portfolio 类

假设用与实现 stock 类相似的方式实现了一系列财产子类集合，然后可以用一个 portfolio 对象列出资产信息。例如，给定下面的财产设置：

```
XYZStock = stock('XYZ', 200, 12);
SaveAccount = savings('Acc # 1234', 2000, 3.2);
Bonds = bond('U.S. Treasury', 1600, 12);
```

创建一个 portfolio 对象：

```
p = portfolio('Gilbert Bates',XYZStock,SaveAccount,Bonds)
```

portfolio 对象的 display 方法可以显示资产内容，如下所示：

```
Descriptor: XYZ
Date: 24-Nov-1998
Current Value:    2400.00
Type: stock
Number of shares: 200
Share price: 12.00
Descriptor: Acc # 1234
Date: 24-Nov-1998
Current Value:    2000.00
Type: savings
Interest Rate: 3.2%
Descriptor: U.S. Treasury
Date: 24-Nov-1998
Current Value:    1600.00
Type: bond
Interest Rate: 12%
Assets for Client: Gilbert Bates
Total Value:    6000.00
```

使用 pie3 方法，用饼图显示各种财产的比例关系。

```
pie3(p)
```

生成图 1-4。

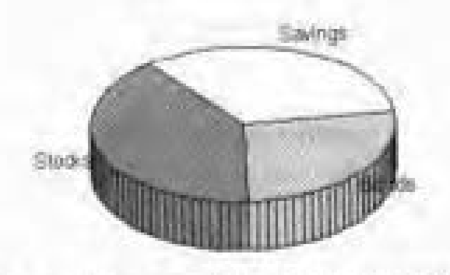


图 1-4 用 portfolio 类的 pie3 方法生成饼图

1.6 保存和装载对象

可以使用 MATLAB 的 save 和 load 命令将自定义对象保存到 .mat 文件或从 .mat 文件中提取出自定义对象。装载对象时，MATLAB 调用对象的构造函数，将对象载入到工作空间。对象类的构造函数必须能在没有输入参数的情况下被调用，并返回默认对象。

1.6.1 保存或载入时修改对象

对对象使用 save 或 load 命令时，MATLAB 在类目录中寻找名为 saveobj 和 loadobj 的类方法。可以在保存或装载以前通过重载这些方法来修改对象。例如，可以定义 saveobj 方法，它保存相关数据和对象，或者编写 loadobj 方法，在对象类型已经载入到 MATLAB 工作空间时，把对象更新到新版本。

1.6.2 示例——为 portfolio 类定义 saveobj 和 loadobj 方法

在上一节的示例中，用 portfolio 类搜集与客户资产有关的信息。现在假设要在保存的每

个 portfolio 对象中添加一个账号，可以定义一个 saveobj 方法，在进行保存操作时自动完成此任务。

进一步假设已保存了很多没有账号的 portfolio 对象，现在要在载入操作时更新它们，并且使其仍然是合法的 portfolio 对象，可以通过为 portfolio 类定义 loadobj 方法来实现这一点。

1. 代码变化综述

要实现账号设定，需要添加或改变下面的函数：

- portfolio：需要修改 portfolio 类的构造函数方法来创建一个新字段 accountNumber，创建对象时它初始化为空字符串。
- saveobj：portfolio 类的新方法。在进行保存操作时，如果 portfolio 对象没有账号，该方法给它添加一个账号。
- loadobj：portfolio 类的新方法。用于更新 portfolio 对象的老版本，它们在添加账号结构字段以前保存。
- subsref：portfolio 类的新方法。利用它对 portfolio 对象进行下标引用。
- getAccountNumber：为 MATLAB 函数，返回由客户名前 3 个字母组成的账号。

2. 新 portfolio 类的行为

添加新方法以后，portfolio 类有下面一些新的行为：

- 为账号包含一个字段。
- 第 1 次保存 portfolio 对象时添加账号。
- 将 portfolio 对象载入到 MATLAB 工作空间时更新它们的老版本。

3. saveobj 方法

不管什么时候 save 命令传递 portfolio 对象，MATLAB 都会查找 portfolio 类的 saveobj 方法。如果@portfolio\saveobj 存在，MATLAB 会将 portfolio 对象传递给 saveobj 方法，然后它必须将修改后的对象作为输出参数返回。下面的代码实现 saveobj 方法，确定对象是否已经在前面的保存操作中指定了一个账号。如果没有，saveobj 函数会调用 getAccountNumber 来获取账号并将它赋给 accountNumber 字段。

```
function b = saveobj(a)
if isempty(a.accountNumber)
    a.accountNumber = getAccountNumber(a);
end
b = a;
```

4. loadobj 方法

不管什么时候 load 命令在正在载入的 .mat 文件中发现 portfolio 对象，MATLAB 都会查找 portfolio 类的 loadobj 方法。如果 loadobj 方法存在，MATLAB 将 portfolio 对象传递给 loadobj 方法，然后它必须将修改后的对象作为输出参数返回，再将输出参数载入到工作空间中。

如果输入对象与构造函数指定的当前定义不匹配，MATLAB 将它转换为一个包含相同字段的结构，并且对象的结构保持原有的值不变。

下面的代码实现 loadobj 方法，首先用 isa 函数确定输入参数是 portfolio 对象还是结构。如果输入是对象，只需要简单地返回它，因为不需要进行修改。如果输入参数已经被 MATLAB 转换为结构，则将新字段 accountNumber 添加给结构，并创建一个更新后的 portfolio 对象。

```
function b = loadobj(a)
% portfolio 类的 loadobj 方法
if isa(a,'portfolio')
      b = a;
else % a 是老版本
      a.accountNumber = getAccountNumber(a);
      b = class(a,'portfolio');
end
```

5. 改变 portfolio 类的构造函数

portfolio 结构数组需要一个额外的字段来包含账号，为了创建该字段，将下面的命令行

```
p.accountNumber = '';
```

添加到@portfolio\portfolio.m 中无参数和有参数的代码段。

6. getAccountNumber 函数

本例中，getAccountNumber 是一个 MATLAB 函数，它返回前面是客户名称的前 3 个字母，后面跟一串数字的账号。getAccountNumber 函数不是一个 portfolio 方法，因此它不能直接访问 portfolio 对象的数据。所以，有必要定义 portfolio 的 subsref 方法来访问 portfolio 对象结构中的名称字段。

本例中，getAccountNumber 函数简单地创建一个随机数，它用 portfolio 名称字段的第 1 到 3 个元素进行格式化和聚合。

```
function n = getAccountNumber(p)
% 为对象 p 提供一个账号
n = [upper(p.name(1:3)) strcat(num2str(round(rand(1,7)*10))')'];
```

注意，portfolio 对象用字段名进行索引，然后用数值下标提取出前 3 个字母。必须编写 subsref 方法来支持该下标引用形式。

7. portfolio 类的 subsref 方法

MATLAB 遇到下标引用，例如在 getAccountNumber 函数中所做的：

```
p.name(1:3)
```

MATLAB 会调用 portfolio 类的 subsref 方法来解释该引用。如果没有定义 subsref 方法，就不会为 portfolio 对象定义上面的语句。

为了能使 getAccountNumber 函数访问 portfolio 类的 name 字段，portfolio 类的 subsref 方法必须支持字段名和数值索引。

```
function b = subsref(p,index)
%为 portfolio 对象定义字段名索引
switch index(1).type
case '.'
      switch index(1).subs
      case 'name'
            if length(index)== 1
                  b = p.name;
            else
                  switch index(2).type
                  case '()'
                        b = p.name(index(2).subs{:});
```

```
            end
        end
    end
end
```

1.7 对象优先级

一般来讲，MATLAB 假设对象有相同的优先级，但下面两种情况例外。

- 自定义类具有比 MATLAB 更高的优先级。
- 使用 inferiorto 和 superiorto 函数，可指定自定义类的相对优先级。

1.7.1 指定自定义类的优先级

通过在构造函数中调用 inferiorto 或 superiorto 函数来指定自定义类的相对优先级。在优先层次中，inferiorto 函数将一个类放在其他类的下面。该函数的调用语法为：

```
inferiorto('class1','class2',…)
```

可以在参数列表中指定多个类，把这些类放在其他类的下面。同样，superiorto 函数将一个类放到优先层次中其他类的上面。该函数的调用语法为：

```
superiorto('class1','class2',…)
```

1.7.2 在优先层次中定位

在优先层次中，如果 objectA 在 objectB 的上面，则表达式 objectA+objectB 调用@classA/plus.m；相反，如果 objectB 在 objectA 的上面，则该表达式调用@classB/plus.m。

一个目录中可能包含许多函数(M 文件)。在单个目录中，函数名必须是惟一的。但多个目录中可以包含名称相同的函数。在命令行中键入函数时，MATLAB 必须搜索可能调用该函数的所有目录。查找函数时，MATLAB 按路径中罗列的顺序搜索目录，并调用名称与指定函数相匹配的第一个函数。面向对象编程允许有多个同名的方法，对于 MATLAB 来说，这些方法就是类目录中的 MATLAB 函数，根据传递给函数的参数所属的类，MATLAB 确定使用什么方法。例如，如果 p 是一个 portfolio 对象，则 pie3(p)调用@portfolio/pie3.m。因为参数是 portfolio 对象。

使用 which 命令，可以确定 MATLAB 将调用哪个方法。例如：

```
which pie3
your_matlab_path/toolbox/matlab/specgraph/pie3.m
```

但是，如果 p 是一个 portfolio 对象：

```
which pie3(p)
dir_on_your_path/@portfolio/pie3.m        % portfolio 方法
```

如果传递一个 portfolio 对象作为输入参数，which 命令确定将调用哪一个 pie3 版本。要察看 MATLAB 路径上特定函数所有版本的列表，使用-all 选项。

第 2 章　改善 MATLAB 的运行效率

与 Visual Basic 一样，MATLAB 是解释型语言，存在计算速度慢的问题。为了提高程序的运行效率，MATLAB 提供了多种实用工具，建议了一些编码技巧。本章将对编码技巧、Profiler 工具、profile 函数和内存处理等进行介绍。

2.1　改善运行的技巧

本节是一个综述性的介绍。内容包括分析程序运行状况的工具、函数和方法，循环矢量化，内存预分配和其他一些技巧。

2.1.1　分析程序的运行状况

通过 Profiler 工具和查看运行时间的函数，能够得到程序如何运行的详细信息，并帮助找到需要改进的地方。

1．Profiler 工具

加速程序运行的第 1 步是找到花费时间比较多的代码行的位置并优化这些代码。使用 Profiler 工具，可以找到这些位置。关于 Profiler，请参见 2.2 节的内容。

2．查看运行时间的函数

如果需要知道程序运行所花费的时间，或者比较不同程序的运行速度，可以使用查看运行时间的函数 tic 和 toc。调用 tic 函数时启动计时器，后面的第 1 个 toc 函数终止它并报告所花费的时间。如：

```
tic
    – 运行要计算时间的程序段
toc
```

3．计算短程序运行所花费的时间

有的程序很短，运行时，因为速度太快，以至于使用 tic 和 toc 都得不到有用的信息。发生这种情况时，尝试把该程序放到一个循环中，然后将执行该循环所花费的时间除以循环次数得到运行那个短程序一次所花费的时间。如：

```
tic;
    for k = 1:100
            – 运行短程序 100 次
    end;
toc
```

则运行短程序一次所花费的时间等于执行整个循环所花费的时间除以 100。

2.1.2 循环矢量化

1. 矢量化方法

MATLAB 是一个矩阵语言，是为矢量和矩阵操作设计的，可以通过矢量化方法来加速 M 文件的运行。矢量化指的是将 for 循环和 while 循环转换为等价的矢量或矩阵操作。例如，下面以 0.01 为间隔计算 0 到 10 之间 1001 个数的正弦值。

```
i = 0;
for t = 0:.01:10
    i = i+1;
    y(i) = sin(t);
end
```

矢量化以后，为：

```
t = 0:.01:10;
y = sin(t);
```

矢量化以后比未矢量化时要快很多，可以用 tic 和 toc 函数进行测试。

2. 使用了矢量化的函数

MATLAB 中，有些函数内部已经采用了矢量化处理，因而运行效率比较高。这些函数的函数名如表 2-1 所示。

表 2-1 MATLAB 中使用了矢量化的函数

all	diff	ipermute	permute	reshape	squeeze
any	find	logical	prod	shiftdim	sub2ind
cumsum	ind2sub	ndgrid	repmat	sort	sum

下面以 repmat 函数为例，具体介绍它内部是如何进行矢量化处理的。该函数有 3 个输入变量：数组 A、行维数 M 和列维数 N，通过将数组 A 做 M×N 的“叠置”操作来生成新的矩阵，如下所示：

```
A = [1 2 3; 4 5 6];
B = repmat(A,2,3);
B =
    1    2    3    1    2    3    1    2    3
    4    5    6    4    5    6    4    5    6
    1    2    3    1    2    3    1    2    3
    4    5    6    4    5    6    4    5    6
```

repmat 函数通过矢量化创建编码，这些编码把元素放在输出数组中。该函数的代码如下所示：

```
function B = repmat(A,M,N)
if nargin < 2
   error('Requires at least 2 inputs.')
elseif nargin == 2
   N = M;
end

% 第 1 步获取行和列的大小
```

```
[m,n] = size(A);

% 第 2 步生成编号从 1 到行或列的大小的矢量
mind = (1:m)';
nind = (1:n)';

% 第 3 步根据上面的矢量创建编号矩阵
mind = mind(:,ones(1,M));
nind = nind(:,ones(1,N));

% 第 4 步创建输出数组
B = A(mind,nind);
```

可见，repmat 函数通过 4 个步骤来生成新矩阵。

第 1 步获取输入数组行和列的大小。

第 2 步创建两个列矢量，mind 包含从 1 到 A 的行数的整数，nind 包含从 1 到 A 的列数的整数。

第 3 步通过一个 MATLAB 矢量化技巧把单列数据复制到任意多列。代码为：

```
B = A(:,ones(1,n_cols))
```

其中，n_cols 是最终生成的矩阵的列数。

第 4 步通过数组索引创建输出数组。行号数组 mind 的每个元素与列号数组 mind 的每个元素相匹配，通过下面的过程来实现：

（1）mind 的第 1 个元素，即行号，与 nind 的每个元素相匹配。MATLAB 按列序在 nind 矩阵中移动，所以 mind(1,1)首先从 nind(1,1)开始，然后是 nind(2,1)。其结果是输出数组的第 1 行被填满。

（2）对 mind 中的每个元素重复上面的过程，得到输出数组。

2.1.3 数组的内存预分配

1．给数组预分配内存

可以通过给保存输出结果的数组预分配内存空间来改善代码的运行效率。预分配使得不必在每次数组变大时进行改变。针对不同类型的数组使用合适的预分配函数如表 2-2 所示。

表 2-2 针对不同类型的数组使用合适的预分配函数

数组类型	预分配函数	示 例
数值数组	zeros	y = zeros(1,100);
单元数组	cell	B = cell(2,3); B{1,3} = 1:3; B{2,2} = 'string';
结构数组	struct repmat	data = repmat(struct('x',[1 3],... 'y',[5 6]), 1, 3);

使用大型矩阵时，预分配还可以帮助减少内存碎片。MATLAB 中，进行动态的内存分配和取消分配时，内存可能会成为碎片。这将导致大量闲置内存的产生。预分配则可以通过在计算以前给大型数据结构“预约”足够的空间来避免这个问题。

2. 给非 double 型矩阵预分配内存

给 double 型以外的矩阵预分配一个内存块时，使用 repmat 函数的效果更佳，此时可以取得更好的效率和更快的运行速度。

下面的语句用 zeros 函数预分配一个 uint8 型 100×100 的矩阵。首先，创建一个 double 型满秩矩阵，然后把矩阵转换为 uint8 型，这将导致不必要的时间和内存花费。

```
A = int8(zeros(100));
```

使用 repmat 函数，只需要创建一个 double 值，从而减少了对内存的需求。

```
A = repmat(int8(0), 100, 100);
```

在不能进行预分配的时候，看是否能通过 repmat 函数使数组变大，用 repmat 函数扩展矩阵时，可以获得连续的内存块。

2.1.4 加速运行的其他方法

1. 用 MEX 文件编写循环代码

必须使用 for 循环时，把它写为 MEX 文件。这样，循环会运行得更快。因为不必在每次运行循环中的语句时都对它们进行解释。

2. 变量赋值

进行变量赋值时，要注意两点。首先，改变已有变量的数据类型或数组形状会使程序运行速度减慢，因为这需要额外的时间进行处理。需要保存不同类型的数据时，建议创建新变量。其次，如果一个变量已经赋给了实数或复数，现在又赋给它复数或实数会影响运行速度。

3. 操作实型数据

MATLAB 为操作实型数特意设计了一些函数，使用这些函数进行实数操作时比使用一般函数快。这些函数包括 reallog,realpow 和 realsqrt 等。

4. 使用合适的逻辑运算符

进行逻辑 AND 或逻辑 OR 时，有两种选择，如表 2-3 所示。

表 2-3 逻辑 AND 和逻辑 OR 的两种选择

运 算 符	描 述
&,|	对数组进行逐元素的逻辑 AND 或逻辑 OR 比较
&&,||	对标量值进行逻辑 AND 或逻辑 OR 比较

在 if 和 while 语句中，使用&&和||更有效率。因为这两个运算符不一定计算整个表达式。例如下面的语句中，只要输入参数的个数小于 3，MATLAB 就会只计算表达式的前半部分，而不会继续计算下去。

```
if (nargin >= 3) && (ischar(varargin{3}))
```

5. 重载内部函数

一般不要重载 MATLAB 内部函数，因为重载内部函数比直接使用内部函数费时。

6. 函数比脚本运行更快

函数中的代码比脚本中的代码运行得更快。在 MATLAB 中，每次使用脚本时都要把它装入内存，然后逐行运行；而函数则被编译为 P 代码，只需要装入内存一次。

7. 使用 load 函数和 save 函数比 MATLAB 文件 I/O 过程更好

使用这两个函数，运行更快，内存碎片更少。

8. 避免大型后台处理

避免在 MATLAB 中运行程序时，在后台进行大型处理，这样，可以为 MATLAB 空出更多 CPU 时间。

2.2 程序运行情况监测——Profiler

Profiler 是一个能够监测程序运行状况的工具，它告诉你 M 文件中哪些代码行最花费时间，哪些行被调用的次数最多。然后，就可以利用上面介绍的方法对这些代码进行改进，从而改进整个程序的性能。

2.2.1 Profiler 的运行环境

在"Start"菜单的"MATLAB"子菜单中单击"Profiler"选项，或在命令窗口中键入 Profile viewer 打开 Profiler 界面，如图 2-1 所示。

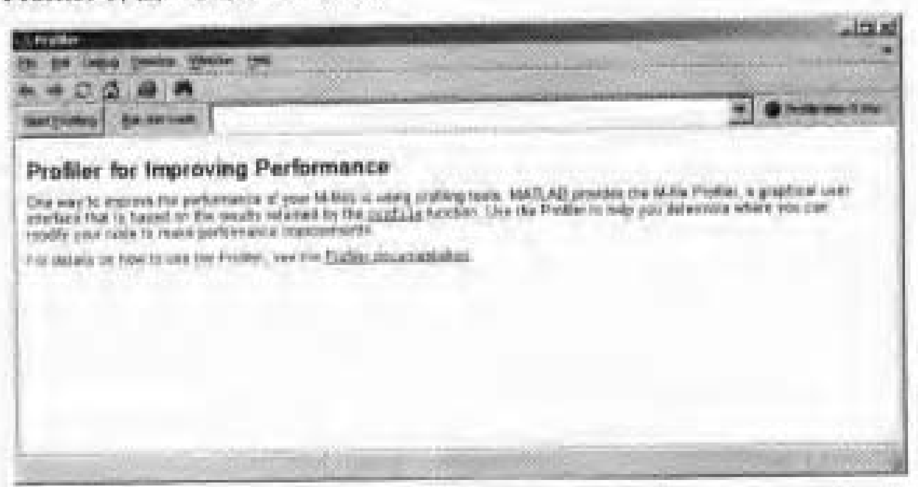


图 2-1 Profiler 的界面

2.2.2 使用 Profiler

按照下面的步骤监测 M 文件或代码行：

（1）在 "Run this code:" 文本框中输入希望运行的语句，例如：

[t,y] = ode23('lotka',[0 2],[20;20])

如果运行前面监测过的语句，如图 2-2 所示，从列表框中选择语句并跳至第 3 步。

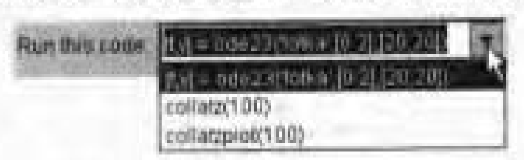


图 2-2 从下拉式列表框中选择语句

（2）单击 "Start Profiling" 按钮，或在输入语句以后按回车键。

当 Profiler 运行时，Profiler 窗口右上角的 "Profile time:" 指示器为绿色，秒数增加。监

测完成以后，“Profile time:”指示器变成黑色，并显示 Profiler 运行的时间长度。注意，这不是语句运行的真实时间，而是单击“Start Profiling”按钮到单击“Stop Profiling”之间的时间。

（3）监测完成以后，“Profile Summary”报表出现在 Profiler 窗口中。

2.2.3 监测图形用户界面的运行情况

可以监测图形用户界面的运行情况，包括已有的图形用户界面和自己设计的图形用户界面。监测图形用户界面按照下面的步骤进行。

（1）在 Profiler 中单击“Start Profiling”按钮。

（2）启动图形用户界面，如果不想在 Profiler 中包含启动过程，启动图形界面以后不要单击“Start Profiling”按钮。

（3）在 Profiler 中单击“Stop Profiling”按钮，显示“Profile Summary”报表，如图 2-3 所示。

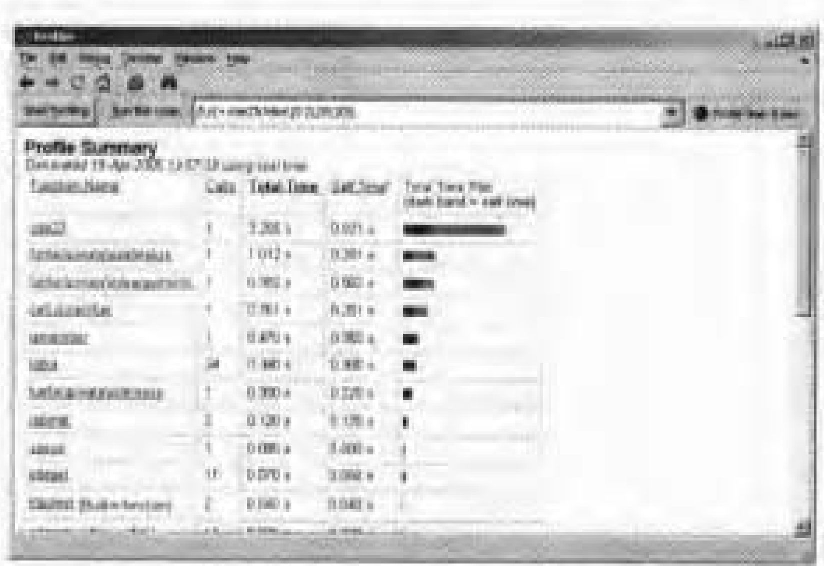


图 2-3 “Profile Summary”报表

2.2.4 从命令窗口监测语句

按照下面的步骤从命令窗口中监测语句：

（1）在 Profiler 中，清除“Run this code”文本框中的内容并单击“Start Profiling”按钮。

（2）在命令窗口中，输入并运行要监测的语句。

（3）运行所有语句以后，在 Profiler 中单击“Stop Profiling”按钮，显示“Profile Summary”报表。

2.2.5 监测综述报表

“Profile Summary”报表提供了运行文件的时间和调用频率。注意，运行时间不连续的文件，其总的时间为 0。

从综述报表中找到需要改进速度的函数，这些函数所花费的时间或调用次数比较多，对于这些函数，在“Function Name”一列中单击函数名，将显示一个详细的报表，如图 2-4 所示。

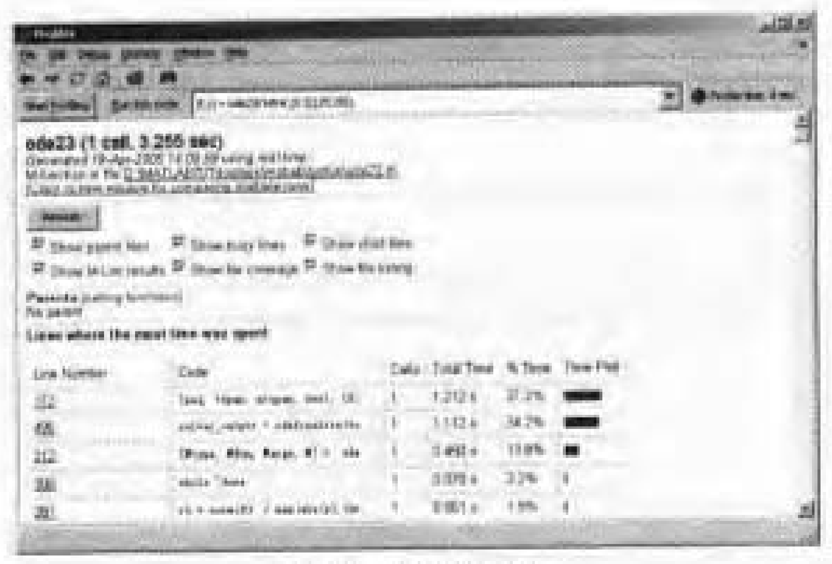


图 2-4　监测详细报表

2.2.6　监测详细报表

监测详细报表显示了在监测过程中被调用的文件的监测结果。有 3 个基本部分可以使用。

第 1 部分提供了文件名、到文件的一个链接以及一个将报表复制到单独窗口的链接。复制报表以后，可以改变文件，为更新的文件运行 Profiler，并比较两次运行的监测详细报表。

第 2 部分提供了包括父函数、花费时间最多的语句行、被子函数调用的语句行的信息的综述列表、M-Lint 结果和文件总信息。

在图 2-4 所示的详细报表中，“Lines where the most time was spent”项目显示文件“ode23.m”运行时花费时间最多的代码行的列表。其中列出了花费时间最多的代码行的行号、代码、调用次数、花费时间、时间百分数和根据时间花费得到的水平条形图。

第 3 部分列出文件，并允许亮显在其他选项中花费时间最多的行。文件列表如图 2-5 所示。



图 2-5　文件列表

可以从文件列表中提取的信息包括行的颜色、颜色亮显和时间、被调用和加速的行。

（1）代码行的颜色

代码行的颜色有不同的含义，具体内容如表 2-4 所示。

表 2-4　代码行的颜色

行的颜色	示　例	描　述
Green	% %ODEARGUMENTS Helper	注释
Black	13 if FcnHandlesUsed	运行的代码行
Gray	14 [illegible]	没有运行的代码行

根据行的颜色可以看出代码行的实际运行状态。这不仅对于 Profile M 文件有用，还可以调试或查看 M 文件如何工作。

（2）颜色亮显

在“Color highlight code according to”下拉式列表框中单击亮显类型以改变外观，本例中选择“Time”，结果如图 2-6 所示。



图 2-6　选择“Time”选项以后的代码行显示

表 2-5 对不同亮显类型与代码行颜色的关系给出了示例和详细描述。

表 2-5　亮显类型与代码行颜色的关系

亮显类型	示　例	描　述
时间	168 if any(tdir*diff(184 f0 = feval(ode,t0, 134 [illegible] 13 if FcnHandlesUsed	阴影色为桃红色的行使用时间最多。其中，比较深的桃红色对应的行花费的时间最多，淡桃红色对应的行占用的时间最少 单击行号，在默认编辑器中打开 M 文件 单击加了下划线的函数查看它的监测详细报表
被调用次数	283 done = false; 284 while ~done	行阴影色为蓝色时调用次数最多。在添加阴影的行中，深蓝色对应行的被调用次数最多，淡蓝色行调用次数最少，行没加阴影但文本为黑色时正在运行，但不在调用最频繁的行中
是否被调用	115 if (length(rtol) ~= 116 error(['RelTol must' 117 end 118 if rtol < 100 * eps	阴影色为蓝色的行被调用，没加阴影的行不被调用。所有加了阴影的行文本用黑色表示，没加阴影的行文本用灰色表示
加速与否	33 else % ode-file used 34 if ~(isempty(options)	加了桃红色阴影的行用 MATLAB 运行加速器加速，没有添加阴影的行不加速
不着色	33 else % ode-file used 34 if ~(isempty(options)	没有行是亮显的

对于M文件的每一行，有3列信息，即时间、调用次数和代码。

2.2.7 利用 Profiler 报表中的信息

1. 改进运行的处理

利用 Profiler 生成的报表，可以改进 M 文件的运行，其方法如下。

（1）在监测综述报表中查找用时最多或调用最频繁的函数。

（2）察看那些函数的详细报表并查找用时最多或调用最频繁的行。

（3）确定是否能对这些行做些改变，以改善运行。例如，循环中有一个 load 语句，每次调用循环时 load 函数都被调用。为了节省时间，可以把 load 语句移到循环的前面，这样只需要调用一次就行了。

（4）单击至文件的链接，进行改进，保存文件并运行“clear all”。重新运行 Profiler 并将结果与开始的报表进行对比。

（5）重复此过程，继续改进代码的运行效率。

2. 使用 Profiler 进行调试

Profiler 也可以用于查找 M 文件中的问题。例如，文件中的一个特定部分没有运行，可以通过在详细报表中查找来发现哪些行在运行，哪些行没有运行，从而发现出问题的点。同样，可以使用“Ctrl+C”键来停止运行，这在文件的运行时间比预期的长得多时很有用。

3. 使用 Profiler 理解 M 文件

对于很长的不是自己创建的 M 文件，或者其他不熟悉的 M 文件，可以使用 Profiler 查看 M 文件是如何工作的。用 Profiler 详细报表查看各行的实际调用情况。

2.2.8 改变 Profiler 的字体

要改变 Profiler 用到的字体，按照以下的步骤进行。

（1）选择 File→Preferences→Help→Fonts，打开“Preferences”对话框。

（2）在该对话框中选择“Font”项，设置字体并单击“Apply”按钮或“OK”按钮。

（3）在 Profiler 中，单击 ▩ 按钮，更新显示。

2.3 使用 profile 函数

Profiler 的运行是基于 profile 函数返回的结果的，但有一些 profile 函数具有的特点在 Profiler 中并没有。

2.3.1 profile 函数的语法和使用步骤

按照下面的步骤使用 profile 函数。

（1）在命令窗口中键入 profile，指定任何想使用的选项。

（2）运行 M 文件，profile 函数计算 M 文件的每一行使用多少时间，它是累加运行的。

表 2-6 是一个 Profile 主窗口的综述表。键入 doc profile，可以查看这些选项以及其他选项的细节。

表 2-6 Profile 主窗口的综述表

语　法	选　项	描　述
profile on		启动 Profile，清除前面记录的统计
	-detaillevel	指定要 Profile 的函数的等级
	-history	指定函数调用的确切次序
profile off		取消 Profile
profile resume		在不清除前面记录的情况下重新启动 Profile
profile clear		清除监测记录的统计信息
profile viewer		打开 Profiler 的图形用户界面，这将提供用 Profile 函数集中的信息，但格式与 Profile 函数报表的不同
s=profile('status')		显示一个包含当前监测状态的结构
stats=profile('info')		延缓监测并显示一个包含监测结果的结构

2.3.2 profile 函数使用演示

下面举一个例子演示如何利用 profile 函数进行运行。

1．启动 Profile

在命令窗口中键入：

```
profile on
```

2．运行 M 文件

本例运行 Lotka-Volterra 猎手 - 猎物人口模型。

```
[t,y] = ode23('lotka',[0 2],[20;20]);
```

3．生成报表并显示

生成监测报表并把结果显示到 Profiler 窗口。

```
profile viewer
```

4．重新启动 Profile

profile 函数现在准备为所运行的更多 M 文件继续搜集统计信息。

```
profile resume
```

5．终止 Profile

```
profile off
```

下面查看监测的结果。有两种察看监测结果的方式，即察看监测报表和监测图。

6．查看监测报表

要显示监测结果，键入：

```
p=profile('info')
p =
        FunctionTable: [23x1 struct]
      FunctionHistory: [2x0 double]
       ClockPrecision: 2.7322e-007
           ClockSpeed: 366
                 Name: 'MATLAB'
```

7．保存报表

用 profsave 函数保存报表。该函数为 p 结构 FunctionTable 字段列出的每个函数将 profile 信息保存在单独的 HTML 文件中。

```
profsave(p)
```

默认时，保存结果放在当前目录下名为 profile_results 的子目录中。

2.3.3 对结果进行访问

监测结果保存在一个可以察看或获取的结构中，本例演示如何察看结果。

1．运行 Profile

在命令窗口键入下面的命令行：

```
profile on -detail builtin -history
[t,y] = ode23('lotka',[0 2],[20;20]);
```

2．查看包含 profile 结果的结构

在命令行键入：

```
stats = profile('info')
```

MATLAB 返回

```
stats =
          FunctionTable: [61x1 struct]
        FunctionHistory: [2x716 double]
         ClockPrecision: 2.7322e-007
             ClockSpeed: 366
                   Name: 'MATLAB'
```

3．察看和获取结构的内容

在命令窗口键入：

```
stats.FunctionTable
```

MATLAB 显示 FunctionTable 结构

```
ans =
61x1 struct array with fields:
    CompleteName
    FunctionName
    FileName
    Type
    NumCalls
    TotalTime
    TotalRecursiveTime
    Children
    Parents
    ExecutedLines
    IsRecursive
    AcceleratorMessages
```

4．查看 FunctionTable 结构中一个元素的内容

在命令行键入：

```
stats.FunctionTable(2)
```

MATLAB 返回结构中的第 2 个元素。

```
ans =
                CompleteName: 'D:\MATLAB701\bin\win32\m_interpreter.dll>horzcat'
                FunctionName: 'horzcat'
                    FileName: 'D:\MATLAB701\bin\win32\m_interpreter.dll'
                        Type: 'Builtin-function'
                    NumCalls: 5
                   TotalTime: 0
          TotalRecursiveTime: 0
                    Children: [0x1 struct]
                     Parents: [2x1 struct]
               ExecutedLines: [0x4 double]
                 IsRecursive: 0
         AcceleratorMessages: {1x0 cell}
```

5. 保存结果

用 save 命令保存结果，即

```
save profstats
```

6. 用保存好的结果生成一个 Profile 报表

在命令行键入：

```
load profstats
profreport(stats)
```

MATLAB 显示 Profile 报表。

2.4 有效使用内存

2.4.1 内存管理函数

使用下面这些函数，可以帮助你在 MATLAB 中管理内存。

（1）whos 函数：显示给工作空间中的变量分配了多少内存。

（2）pack 函数：把已经存在的变量保存到磁盘，然后重新装入，这将减少因为内存碎片出问题的机会。

参见下一小节中“把数据压入内存”的内容。

（3）clear 函数：从内存中删除变量，增加可用内存的一种方法是周期性地把不再使用的变量从内存中清除出去。

（4）save 函数：有选择地把变量保存到磁盘。使用大量数据时，这是一个有用的技巧。

（5）load 函数：把已保存的数据文件用 save 函数重新载入。

（6）quit 函数：退出 MATLAB 并返回所有分配的内存到系统中。

2.4.2 节约内存的方法

本节介绍帮助你更节约地使用内存，并避免“Out of Memory”错误出现的方法。

1．使用变量

为了在创建变量时节约内存，注意：

- 避免创建大型临时性变量，不再需要时清除它们；
- 使用固定大小的数组时，尽量进行预分配；
- 将变量设置为空矩阵[ ]，将它从内存中清除，或用 clear 函数清除变量；
- 尽量重用变量。

2．把数据压入内存

因为 MATLAB 用堆管理内存，所以运行时会产生内存碎片，使内存产生很多闲置的空间。当闲置空间太多时，就没有足够的内存保存新的大型变量，并导致“Out of memory”错误产生。此时，可以用 pack 函数把数据压入内存，从而把更多相邻的内存块“释放”出来。

注意：从运行时间上考虑，不要在循环或 M 文件函数中使用 pack 函数。

3．从内存中清除不再使用的变量

如果使用 pack 函数以后，内存仍然不够，可能需要从内存中删除一些不再使用的变量。

4．有大量数据的情况

如果程序生成大量数据，建议周期性地把数据写入磁盘。保存一部分数据以后，从内存中清除变量并继续生成数据。

5．把满秩矩阵转换为稀疏矩阵

大部分元素为 0 的矩阵最好保存为稀疏矩阵。稀疏矩阵使用的内存更少，并且比满秩矩阵运行更快，可用 sparse 函数进行转换。

比较两个 1000×1000 的矩阵 X 和 Y。其中，X 为 double 型，其中 2/3 的元素等于 0；Y 为 X 的稀疏矩阵形式，如下所示。Y 所占用的内存大约只有 X 的一半。

```
whos
  Name      Size                 Bytes   Class
  X         1000x1000          8000000   double array
  Y         1000x1000          4004000   double array (sparse)
```

6．单元数组的内存需求

整个的单元数组或结构数组，并不需要连续的内存。因为它们实际上是一个指向其他数组的指针数组，要求每个数组的内存是连续的，但整个内存集合则不要求。

7．比较数组结构和结构数组

如果 MATLAB 应用程序需要保存一大套数据，并且数据可以保存为数组结构或结构数组，但前者更好。对于同一套数据，数组结构需要的内存比结构数组少得多，并且处理起来速度也更快。

8．嵌套函数调用

嵌套函数使用的内存与连续行方式调用它们时所占用的内存相同。如下面两个例子需要相同大小的内存。

```
result = function2(function1(input99));
result = function1(input99);
result = function2(result);
```

第3章 编 译 器

编译是 MATLAB 的主要难题之一，一直都不顺畅。MATLAB 7.0 使用新的编译器 4.0，提供了多种新的编译特性，弥补了原有编译器的一些缺点，在漫长的编译之路上前进了一步。

3.1 概述

MATLAB 编译器 4.0 可以用 M 文件等生成可重复分发的独立应用程序或组件，它支持面向对象编程时创建的类对象类型。

3.1.1 MATLAB 编译器 4.0 和以前版本的区别

编译器 4.0 是创建可分发给其他用户的部件的部署工具。这个版本的 MATLAB 编译器完全支持 MATLAB 语言的所有特性，包括面向对象编程定义的对象。

- 编译器 4.0 用新的 MATLAB 组件运行时(Component Runtime，MCR)代替了 MATLAB C/C++数学和图形库。MCR 是共享库组成的独立集合，这些共享库使得可以执行 MATLAB 编译器创建的加密 M 文件。
- 编译器 4.0 只生成接口函数（打包器）的代码，而以前版本的编译器生成整个 M 文件的代码。
- 编译器 4.0 取消了某些与代码生成和格式化有关的选项。
- 编译器 4.0 取消了某些打包器选项和与它们有关的束文件。
- 只有 Microsoft Windows 和 Linux 平台支持编译器 4.0。
- 编译器 4.0 包含了几个新的选项。
- 编译器 4.0 不包含 MATLAB 对 Visual Studio 的插件（Add-in）功能。
- 编译器 4.0 不加速程序。编译后的应用程序与在 MATLAB 上运行没有速度上的差异。编译后的应用程序运行时将会像有 JIT 加速器时一样快。
- 编译器 4.0 不支持源于 M 函数的 MEX 文件和 Simulink S 函数的编译，因为 MATLAB 7 的特性使得这项功能多余。MATLAB JIT 加速器使得编译时不必考虑速度问题，并且使用 MATLAB 的 pcode 函数可以隐藏自己的算法。
- MATLAB 不支持通过 loadlibrary 函数载入 MATLAB 编译器生成的库的功能。

1．打包器编码差异

打包器或打包器文件包含了 MATLAB 编译器生成的代码与支持的部件类型，如独立应用程序或共享库等之间的必要接口。编译器 4.0 只生成接口函数的代码，而以前版本的编译器会生成整个 M 文件的代码。从 C 或 C++函数中调用编译器 4.0 生成的函数时有下面几个需要注意的差异：

- 因为编译器 4.0 不使用 MATLAB C/C++数学和图形库，所以现在得不到一系列 mlf 函数。
- 现在 initialize 函数会返回一个状态标记，它可以测试库是否进行了正确初始化。

在这个版本的编译器中，用 M 文件函数生成的 mlf 函数的接口发生了改变。与以前各版本的 MATLAB 编译器不同，现在所有返回值都作为输入参数传递给该函数。这些函数的返回值为 void。输出的 mlf 函数的一般形式为：

对于没有返回值的 M 函数

void mlf<function-name>(<list_of_input_variables>);

对于至少有一个返回值的 M 函数

void mlf<function-name>(<int number_of_return_values>,
<list_of_pointer_to_return_variables>,
<list_of_input_variables>);

注意：这些打包器文件的差异只影响生成库的用户，对于生成独立应用的用户没有影响。

2. 取消的编译选项

编译器 4.0 比以前的版本更容易使用，而且是一个更有力的工具。以前版本中的某些选项现在不必要了，为了简化编译器，编译器 4.0 取消了这些选项，如表 3-1 所示。

表 3-1 编译器 4.0 取消的选项

选　项	描　述	选　项	描　述
A	代码注释	O	优化后的代码
B pcode	生成 P 代码	P	生成 C++代码
F	格式化参数	S	生成 Sumulink S 函数的宏
h	帮助器函数	t	将 M 代码转换为 C/C++代码
i	包含指定的 M 文件	u	指定 Simulink S 函数输入参数的个数
l	行/文件编号（该选项现在表示"library"，库）	x	生成 MEX 函数的宏
L	目标语言	y	指定 Simulink S 函数的输出参数的个数

3. 取消的打包器选项

表 3-2 中的打包器选项和与它们相关的束文件已经取消，其功能已经被新选项替代。

表 3-2 取消的打包器选项及其替代选项

打包器选项/束文件	用下面的选项替代
B csglcom	B ccom
B csglexcel	B cexcel
B csglsharedlib	B csharedlib
B cppsglcom	B cppcom
B cppsglexcel	B cppexcel
W comhg	W com
W excelhg	W excel
W libhg	W lib
W mainhg	W main

注意：现在不再需要用–B sgl和–B sglcpp 来获取句柄图形函数了，默认时所有经过编译的应用程序都有权使用图形。

4. 新的编译器选项

表 3-3 列出了编译器 4.0 中的新选项。

表 3-3　编译器 4.0 的新选项

选　项	描　述
a filename	将 filename 文件添加到 CTF 文档
l	生成函数库的宏
N	清除必需的最小目录集以外的所有路径
P <directory>	将 directory 目录按照次序敏感上下文添加到编译路径，需要–N 选项
R –nojvm R –nojit	运行时覆盖 MCR 选项

● 添加文件到 CTF 文档：使用–a 选项指定直接添加到 CTF 文档的文件。编译器在 MATLAB 路径上查找这些文件，所以指定完整路径名是可选的。这些文件不传递给 mbuild，所以可以包含数据文件之类的文件。例如，

mcc –m foo.m –a data.mat –a/docs/help.txt

可以使用多个–a 选项，但是每个的后面必须跟要添加到 CTF 文档的文件的名称。可以将–a 选项添加到 mcc 命令行的任何地方。出现在 mcc 命令行的.c, .cpp 或 .obj 等不是 MATLAB 文件的文件直接传递给 mbuild。

● 生成库：使用–l 宏生成函数库。–l 选项等价于：

–W lib –T link:lib

它为命令行的每个 M 文件生成库打包器函数，并调用 C 编译器生成共享库，该共享库输出这些函数。库名与组件名相同，也就是命令行中第一个 M 文件的名称。

● 清除最小目录集以外的路径：使用–N 选项清除下面必要路径以外的所有路径。

<matlabroot>\toolbox\matlab

<matlabroot>\toolbox\local

<matlabroot>\toolbox\compiler

该选项还保留上面列表中编译时出现在 MATLAB 路径中的目录的子目录。在 mcc 命令行包含–N 选项，使得可以替换原始路径，但保留所包含的目录的相对顺序。另外还包含出现在原始路径中的所包含的目录的所有子目录。

● 向编译路径添加目录：使用–P 选项按照次序敏感上下文向编译路径添加目录。语法格式为

p <directory>

其中，<directory>为要包含的目录。如果<directory>不是作为绝对路径传递，则假设位于当前工作目录下。

● 运行时：使用–R 选项覆盖 MCR 运行时选项。可以覆盖表 3-4 中的任何运行时选项。默认时这些选项的值为 on。

表 3-4　运行时选项

MCR 选项	描　述
nojvm	不使用 Java 虚拟机（JVM）
nojit	不使用 JIT 加速器（用于加速 M 文件执行速度的二进制代码生成）

下面是–R 选项的合法用法：

mcc –m –R "–nojvm <args> –nojit <args>" –v foo.m

```
mcc –m –R "–nojvm <args>" –v –R "–nojit <args>" foo.m
mcc –m –R –nojvm –R –nojit foo.m
mcc –m –R –nojvm –v foo.m
mcc –m –R nojvm –R nojit foo.m
```

下面的用法是非法的：

```
mcc –m –R –nojvm –nojit foo.m
mcc –m –R –nojvm <args> –R –nojit <args> foo.m
```

3.1.2 MATLAB 编译器的基本功能

MATLAB 编译器可以生成独立应用程序、库、COM 对象和 Excel 插件。

1. 打包器文件

MATLAB 编译器（mcc）根据 M 文件生成可以分发的独立应用程序或软件组件。最终生成的结果可以是独立应用的库或组件中的任何一种类型。编译器根据最后要生成的目标类型生成合适的打包器文件。打包器文件包含编译后的应用程序和所支持的可执行对象类型之间的接口。

打包器文件根据执行环境的不同而有所不同。为了提供必要的接口，打包器具有下面一些功能：

- 完成打包器指定的初始化和终止运行工作；
- 定义包含路径信息、加密密匙和 MATLAB 组件运行时（MCR）所需要的其他信息的数组；
- 提供传递接口函数调用 MCR 中 MATLAB 函数时的必要代码。

例如，打包器是一个包含 main 函数的独立应用。库的打包器包含每个公共 M 文件函数的入口点。

MATLAB 编译器生成的组件技术文件（CTF）与最后生成的目标类型（可执行程序或库）是独立的。打包器文件会提供与目标类型之间的必要接口。

2. 独立应用

用–m 选项调用时，MATLAB 编译器使用输入的 M 文件，生成适合于独立应用的必要的打包器文件。然后，C 或 C++ 编译器编译该代码并连接 MCR。例如，下面的代码生成文件 example.m 的独立可执行文件

```
mcc –m example
```

3. 库

可以用–l 选项根据 M 文件集生成 C 共享库。例如，

```
mcc –l file1.m file2.m file3.m
```

用 file1.m,file2.m 和 file3.m 生成一个共享库。–l 选项是一个打包选项，可以扩展为

```
–w lib –T link:lib
```

–w lib 选项让 MATLAB 编译器为共享库生成一个函数打包器，并调用它 libfile1。–T link:lib 选项把目标输出指定为共享库。

4. COM 生成器和 Excel 生成器

使用可选的 MATLAB COM 生成器，可以创建 COM 组件。该组件可以用于任何操作 COM

对象的应用程序。MATLAB COM 生成器用编译器根据 M 文件创建组件对象模型（COM），M 文件集合被转换为 COM 类。COM 生成器支持一个组件中有多个类的情况。

利用可选的 Excel 生成器，可以自动生成 Visual Basic 应用文件（.bas）和插件 DLL。可以在 Excel 中直接使用 .bas 文件和导入插件。MATLAB Excel 生成器将 MATLAB M 文件编译为可以用作 Excel 插件的 COM 对象，M 文件集合被转换为单个的 Excel 插件。MATLAB Excel 生成器支持一个组件中有一个类。

3.1.3 使用 MATLAB 编译器的基本步骤

下面介绍使用 MATLAB 编译器用 M 文件创建部件（应用程序和库）的基本步骤。然后这些部件可以分发到那些没有安装 MATLAB 的机器上。

1．编译独立应用程序

用 M 文件 mymfunction 创建独立应用程序，使用下面的命令

```
mcc –m mymfunction.m
```

创建一个名为 mymfunction.exe 的独立可执行程序。

2．编译共享库

用 M 文件 mymfunction 创建共享库，使用下面的命令

```
mcc –l mymfunction.m
```

创建一个名为 mymfunction.dll 的共享库。

3．调用库的独立应用程序

假设名为 mywrappercode 的打包器文件调用上面的库 mymfunction，现在用该打包器文件创建独立（驱动）程序，使用下面的命令

```
mbuild mywrappercode.c mymfunction.lib
```

4．在开发机器上测试部件

在开发机器上测试部件，必须保证动态链接库设置正确。首先添加下面的目录到动态链接库路径中。

```
<matlabroot>\bin\win32
```

然后可以在开发机器上运行编译后的应用程序，进行测试。

5．把部件部署到其他机器上

将部件部署到没有安装相同版本的 MATLAB 的目标机器上，需要将下面的部件打包并注册目标机器。

- 部件，即独立可执行程序或共享库。
- 为部件创建的 CTF 文档(<component_name>.ctf)。
- MCRInstaller.exe(该文件位于<matlabroot>\toolbox\compiler\deploy\win32 目录下)。

在目标机器上完成以下操作：

- 通过运行某目录下的 MCR 安装器来安装 MCR。例如，运行 C:\MCR 目录下的 MCRInstaller.exe。
- 将部件和 CTF 文档复制到应用程序根目录，例如 C:\approot。
- 将下面的目录添加到系统路径。

<mcr_root>\runtime\win32

- 测试部件。

3.1.4 MATLAB 编译器的局限性

1．编译 MATLAB 和工具箱

MATLAB 编译器支持完整的 MATLAB 语法和几乎所有基于 MATLAB 的工具箱。但是，有些 MATLAB 和工具箱功能不能编译，包括：

- 大部分预生成的图形用户界面及相关工具箱；
- 不能直接从命令行调用的函数；
- 有些工具箱，例如符号数学工具箱。

2．MATLAB 编码

MATLAB 编译器支持 MATLAB 的绝大部分功能，但是，有一些局限性需要注意。编译器的这个版本不能创建脚本式 M 文件的接口。

MATLAB 技术支持通过互联网随时发布新版本的 M 文件来修改程序中的 bug。将这些改变集成到部署的应用程序中，必须首先使用补丁，然后重新运行 buildmcr 来生成最新的 MCRInstaller 版本。向客户部署 bug 修复工具，必须与新应用程序一起发行这个新的 MCRInstaller，并使当前客户可以得到该安装器，以便更新其安装。

3．独立应用

前面提到的限制对于独立应用来说也适用。另外，有一些函数不支持独立模式，主要有下面几类：

- 打印或报告 MATLAB 函数代码的函数，例如 help 函数和一些调试函数。
- Sumulink 函数。
- 需要在命令行运行的函数，例如 lookfor 函数等。
- clc、home 和 savepath 等函数部署以后不再有效。

独立应用成功运行时返回值为 0，否则返回值为非 0。

另外，由于授权限制，有一些函数是不能部署的，如表 3-5 所示。

表 3-5　不支持的函数

add_block	add_line	applescript	close_system	dbclear	dbcont
dbdown	dbquit	dbstack	dbstatus	dbstep	dbstop
dbtype	dbup	delete_block	delete_line	echo	edit
fields	get_param	help	home	inmem	keyboard
linmod	mislocked	mlock	more	munlock	new_system
open_system	pack	rehash	set_param	sim	simget
simset	sldebug	type			

4．固定回调问题：缺失函数

编译器创建独立应用时，会编译命令行中指定的 M 文件，另外，还编译该 M 文件调用的任何其他 M 文件。编译器使用依赖分析来确定指定 M 文件、MEX 文件和 P 文件所依赖的函数。如果函数在下面几个位置调用，依赖分析将找不到该函数。

- 在回调字符串中；

- 在作为参数传递给 feval 函数或 ODE 求解器的字符串中。

编译器不会在这些文本字符串中查找函数名进行编译。

出现这种问题时，程序可以运行，但图形界面上的某些元素，例如命令按钮，可能不能工作。编译后的程序发出下面的出错消息。

```
An error occurred in the callback : change_colormap
The error message caught was   : Reference to unknown function
     change_colormap from FEVAL in stand-alone mode.
```

要清除这个错误，需要创建一个只在回调字符串中指定的所有函数组成的列表，并用单独的%#function 注记语句传递这些函数。这将覆盖编译器的依赖分析，并使它可以显式地包含%#function 注记中列出的函数。

例如，在示例程序 my_test 中调用 change_colormap 函数。要确保编译器处理 change_colormap M 文件，需要用%#function 注记语句列出函数名。

```
function my_test()
% Graphics library callback test application
%#function change_colormap
peaks;
p_btn = uicontrol(gcf,...
          Style', 'pushbutton',...
          Position',[10 10 133 25 ],...
          String', 'Make Black & White',...
          CallBack','change_colormap');
```

注意：*如果不使用%#function 注记语句，可以在编译器命令行中用–a 选项指定缺失的 M 文件的名称。*

那么程序中的哪些函数会出现在%#function 注记语句中呢？需要在 M 文件的源代码中进行搜索。搜索那些作为回调字符串或 feval、fminbnd、fminsearch、funm 和 fzero 函数或任何 ODE 求解器的参数的文本字符串。

查找用作回调字符串的文本时，在 M 文件中搜索字符"Callback"和"fcn"，这将找到句柄图形对象如 uicontrol 和 uimenu 等定义的所有 Callback 属性。另外，将找到图形窗口对象或坐标系对象所有以 Fcn 结尾的属性，例如 CloseRequestFcn 等，它同样支持回调。

3.1.5 关于运行时服务器(Runtime Server)

6.5 及以前版本的 MATLAB 中，除了提供编译器外，还提供了运行时服务器进行程序编译。运行时服务器的使用比较烦琐，而且很多情况下不太顺畅。在 MATLAB 7.0 中，运行时服务器的大部分功能集成到编译器 4.0 中去了，所以 Mathworks 公司不再把它作为单独的产品进行提供。

3.2 安装和注册

本章介绍 MATLAB 编译器的系统需求，以及安装和注册信息。安装 ANSI C 或 C++编译器时，可能需要提供指定的注册细节。

3.2.1 系统需求

安装 MATLAB 编译器 4.0，必须首先安装 MATLAB 7。MATLAB 编译器对操作系统和内存没有额外的要求，只需要很少的磁盘空间。

MATLAB 编译器要求首先安装一个它支持的 ANSI C 或 C++编译器，某些特定的输出要求特殊的编译器。

通常，MATLAB 编译器支持第 3 方编译器的当前版本和以前诸版本。通过下面的网页可查看 MATLAB 和 MATLAB 编译器支持的所有编译器。

http://www.mathworks.com/support/tech-notes/1600/1601.shtml

可以使用下面 32 位 C/C++编译器中的任何一个来创建 32 位 Windows 动态链接库或 Windows NT 应用程序。

- Lcc C 2.4（MATLAB 已经包含该编译器）：C 编译器，不能用于 C++的编译。
- Borland C++ 5.3,5.4,5.5,5.6（有的称 Borland C++ Builder 3.0,4.0,5.0,6.0）
- Microsoft Visual C/C++(MSVC) 6.0,7.0 和 7.1

3.2.2 安装

1. 安装 MATLAB 编译器

如果有安装 MATLAB 编译器的授权，则安装 MATLAB 时会有是否安装编译器的选项，选择它就可以安装。

如果安装 MATLAB 时没有出现是否安装编译器的选项，需要跟 MathWorks 公司联系。多用户网络安装时索取更新授权文件(license.dat)，单机标准安装时索取更新个人授权密码(PLP)。

2. 安装 ANSI C/C++编译器

安装 ANSI C/C++编译器时，按照厂商的提示进行安装。必须对 C/C++编译器进行测试，确保已经正确安装并注册。一般，厂商会提供一些测试程序。

安装 C/C++编译器时，可能会遇到需要提供特定细节的注册问题。表 3-6 提供了某些更常见主题的信息。

表 3-6 注册常见问题

问　题	说　明
安装选项	建议完整安装编译器。如果只安装一部分，可能会忽略 MATLAB 编译器必需的组件
安装调试文件	如果只是为了进行编译，不必安装调试器文件。但是，在其他某些情况下需要这些文件
MFC	不是必需的
16 位 DLL/可执行程序	不是必需的
ActiveX	不是必需的
从命令行运行	确定选择了从命令行运行编译器的所有相关选项
更新注册	如果安装器给出了更新注册信息的选项，更新它
安装 Microsoft Visual C/C++ 6.0	如果需要改变编译器的安装位置，必须改变 Common 目录的位置。不要改变 VC98 目录的位置

3.2.3 注册

MATLAB 有一个名为 mbuild 的工具，它可以简化 C/C++编译器的安装过程。一般情况下，只需要用 mbuild 工具的 setup 选项来指定使用哪个第 3 方编译器。

1. mbuild 工具简介

使用 mbuild 工具可以定制注册和生成过程。mbuild 脚本提供了指定选项文件的简单方式，可以实现：

- 编译器和链接器设置；
- 改变编译器或改变编译器的设置；
- 生成应用程序。

MATLAB 编译器（mcc）在某些条件下会自动调用 mbuild 工具。特别地，mcc –m 或 mcc –l 调用 mbuild 进行编译和链接。

2. 注册 ANSI C/C++编译器

编译器的选项文件包含一些标记和设置，它们控制所安装的 C 和 C++编译器的操作。选项文件根据编译器有所不同，即 MATLAB 为支持的每个 C/C++编译器提供惟一的选项文件。选择一款编译器与 MATLAB 编译器一起使用时，对应的选项文件会在系统上激活。选择默认的编译器，使用下面的语句：

```
mbuild -setup
```

本小节还提供了与选项文件有关的其他信息，有些用户可能会根据需要修改它们。大部分用户不必关心选项文件的内部操作，只需要用 setup 选项指定 C/C++编译器就行了。

执行下面的命令：

```
mbuild -setup
```

按照提示指定要使用的编译器。

```
Please choose your compiler for building standalone MATLAB
applications:

Would you like mbuild to locate installed compilers [y]/n? n

Select a compiler:
[1] Borland C++Builder version 6.0
[2] Borland C++Builder version 5.0
[3] Borland C++Builder version 4.0
[4] Borland C++Builder version 3.0
[5] Borland C/C++ version 5.02
[6] Borland C/C++ version 5.0
[7] Borland C/C++ (free command line tools) version 5.5
[8] Lcc C version 2.4
[9] Microsoft Visual C/C++ version 7.1
[10] Microsoft Visual C/C++ version 7.0
[11] Microsoft Visual C/C++ version 6.0

[0] None
```

```
Compiler: 11

Your machine has a Microsoft Visual C/C++ compiler located at
D:\Applications\Microsoft Visual Studio. Do you want to use this
compiler [y]/n? y

Please verify your choices:

Compiler: Microsoft Visual C/C++ 6.0
Location: D:\Applications\Microsoft Visual Studio

Are these correct?([y]/n): y

Try to update options file:
C:\WINNT\Profiles\username\Application
Data\MathWorks\MATLAB\R14\compopts.bat
From template:
\\sys\MATLAB\BIN\WIN32\mbuildopts\msvc60compp.bat

Done . . .
 .
 .
Updated ...
```

表 3-7 显示了 MATLAB 包含的选项文件。

表 3-7 选项文件

选项文件	编译器
lcccompp.bt	Lcc C 2.4
msvc60compp.bat	Microsoft Visual C/C++ 6.0
msvc70compp.bat	Microsoft Visual C/C++ 7.0
msvc71compp.bat	Microsoft Visual C/C++ 7.1
bcc53compp.bat	Borland C++ Builder 3
bcc54compp.bat	Borland C++ Builder 4
bcc55compp.bat	Borland C++ Builder 5
bcc56compp.bat	Borland C++ Builder 6

3.2.4 几个问题

1. Windows 编译器的局限性

对于支持的编译器，有下面几个已知的局限性。

- Lcc C 编译器不支持 C++。
- 只有 Borland C++ Builder(3.0,4.0,5.0 和 6.0)和 Microsoft Visual C/C++(6.0,7.0 和 7.1)支持 COM 对象的生成。用 Borland C++ Builder 生成 COM 对象时需要微软提供的 MIDL 编译器。
- Borland C++编译器有一个局限性。在 M 代码中，如果使用了以 0 打头的常数，并且该常数的小数点前面包含有数字 8 或 9，Borland 编译器会显示下面的出错信息：

```
Error <file>.c <line>: Illegal octal digit in function
```

<functionname>

2. 选项文件

如果想知道选项文件是如何工作的，请阅读下面的内容。

在 Windows 中查找选项文件，mbuild 脚本按照下面的顺序进行查找：

- 当前目录；
- user profile 目录。

mbuild 使用它找到的第 1 个选项文件。如果没有找到选项文件，mbuild 会在机器中进行搜索，查找支持的 C 编译器，并给该编译器使用默认的选项文件。如果找到多个编译器，则提示你从中间选择一个。

Windows 的 user profile 目录包含用户指定信息，例如桌面外观、最近使用过的文件和“开始”菜单中的选项等。mbuild 工具保存它的选项文件 compopts.bat，它用-setup 选项创建，位于 user profile 目录的子目录中，名称为 Application Data\MathWorks\MATLAB\R14。在用户概况可用的 Windows 系统上，user profile 目录为%windir%\Profiles\username；在用户概况不可用的 Windows 系统上，user profile 目录为%windir%。可以用“用户和密码”控制面板确定用户概况是否可用。

3. 改变选项文件

尽管一般情况下对于所有与编译器有关的操作只要使用一个选项文件就行了，但可以随时改变选项文件。setup 选项重新设置默认编译器，使得可以随时使用新编译器。使用下面的命令重新设置 C/C++编译器。

```
mbuild -setup
```

setup 选项将合适的选项文件复制到 user profile 目录下。按照下面的步骤修改选项文件。

- 用 mbuild -setup 命令将合适的选项文件复制到指定目录。
- 编辑 user profile 目录下的选项文件复制，使它合乎要求，并保存修改后的文件。

完成以后，mbuild 脚本会使用新的选项文件。

3.3 编译处理

本节介绍如何使用编译器，列出了编译器用到的几种输入、输出文件集。

3.3.1 MATLAB 编译器术语简介

1. MATLAB 组件运行时

MATLAB 编译器 4.0 使用 MATLAB 组件运行时（MCR），它是能够执行 M 文件的共享库组成的独立集合。MCR 对 MATLAB 语言的所有特性提供了完整的支持。

2. 组件技术文件

编译器 4.0 还使用组件技术文件（Couponent Technology File,CTF）来存放可部署的包。所有 M 文件都加密保存在 CTF 文档中。MATLAB 编译器生成的每个应用程序或共享库都有一个相关的 CTF 文档。文档中包含所有基于 MATLAB 的可执行内容（M 文件、MEX 文件等）。

多 CTF 文档，例如 COM 或 Excel 组件，可以在同一用户程序中共存，但是不能混合和匹配它们所包含的 M 文件。不能将经过加密和压缩的 M 文件从多 CTF 文档中组合到另一个

CTF 文档中并发布它们。

给定 CTF 文档的所有 M 文件都用一个惟一的密匙锁在一起。具有不同密匙的 M 文件放在同一个 CTF 文档中时不能执行。如果试图用 M 文件的不同组合生成另一个应用程序，必须将这些 M 文件重新编译到一个新的 CTF 文档中。

3. 生成过程

用 MATLAB 编译器创建软件组件的过程完全是自动的。例如，要创建独立 MATLAB 应用程序，需要提供组成程序的 M 文件列表，然后编译器进行以下操作：

- 依赖分析
- 生成代码
- 创建文档
- 编译
- 链接

图 3-1 显示了编译器生成独立可执行程序的过程。

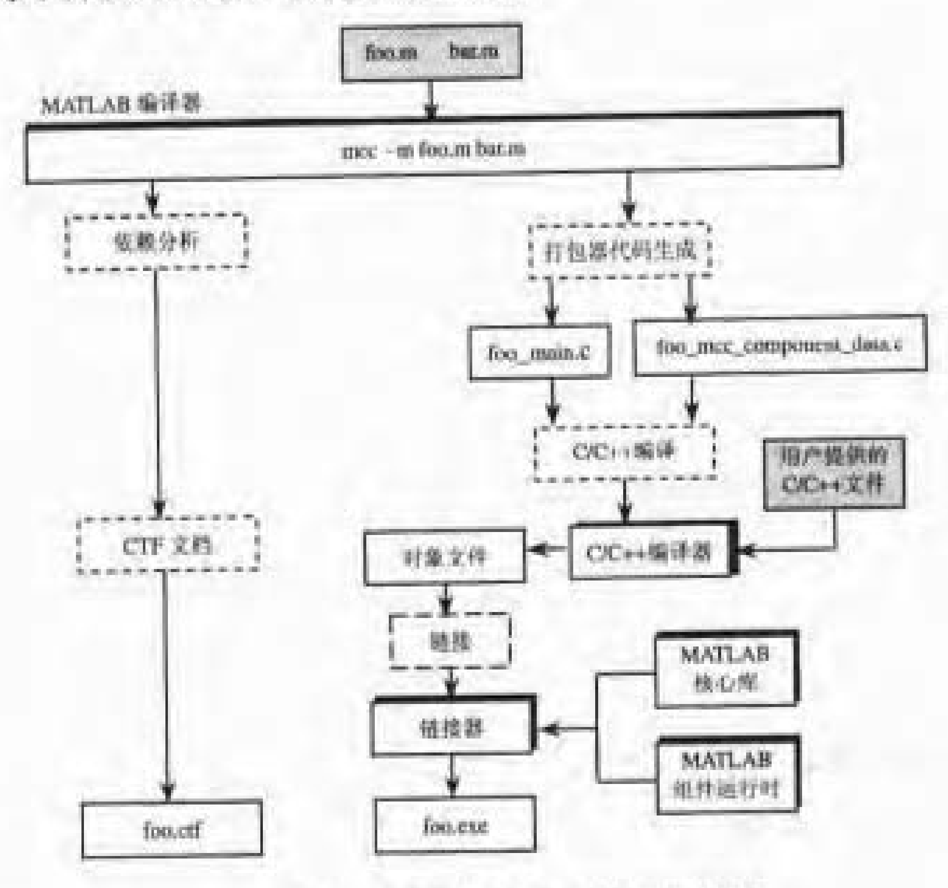


图 3-1 编译器生成独立可执行程序的过程

下面分步进行介绍。

（1）依赖分析

这一步确定指定 M 文件、MEX 文件和 P 文件所依赖的所有函数。列表包括给定文件调用的所有 M 文件及它们调用的所有文件。还包括所有内部函数和 MATLAB 对象。

（2）生成打包器代码

这一步生成创建目标组件所需要的所有源代码，包括：

● 命令行中提供的这些 M 函数的 C/C++接口代码。对于库和组件，该文件包含生成的所有接口函数。

● 组件数据文件包含运行时执行 M 代码所必需的信息。该数据包括路径信息和载入保存在组件 CTF 文档中的 M 代码所必需的密匙。

（3）创建 CTF 文档

依赖分析中创建的 MATLAB M 文件和 MEX 文件列表用于生成 CTF 文档。该文档包含运行时组件正确执行所需要的所有文件。这些文件被加密和压缩到一个单独的文件中进行部署，还包含了目录信息。这样就把所有内容正确安装到了目标机器上。

（4）C/C++编译

这一步将打包器代码生成阶段生成的 C/C++文件编译为结果代码（object code）。为了支持在 mcc 命令行包含用户提供的 C/C++代码，这部分代码也在这个阶段编译。

（5）链接

最后一步，链接生成的结果文件和必要的 MATLAB 库，以创建最后的组件。

C/C++编译和链接都使用 mbuild 工具。

3.3.2 输入和输出文件

本小节描述编译过程中创建的文件。

1. 独立可执行程序

下面的语句中，MATLAB 编译器将 M 文件 foo.m 和 bar.m 作为输入，生成独立可执行程序 foo.exe。

```
mcc –m foo.m bar.m
```

输出文件如表 3-8 中所示。

表 3-8 输出文件

文　件	描　述
foo_main.c	主打包器源文件，包含程序的 main 函数。main 函数将输入参数作为字符串传递给 foo 函数
foo_mcc_component_data.c	C 源文件，包含 MCR 运行程序所必需的数据。该数据包含路径信息、密匙和其他初始化信息
foo.ctf	CTF 文档，该文件包含组成应用程序的 M 文件（foo.m 和 bar.m）的压缩加密文档，还包含两个主 M 文件调用的其他文件，以及运行时需要的所有其他可执行内容和数据文件
foo.exe	应用程序的主可执行文件。该文件读取和执行保存在 CTF 文档中的内容

2. C 共享库

本例中，输入为 M 文件 foo.m 和 bar.m，生成 C 共享库 libfoo.dll。使用下面的命令：

```
mcc –W lib:libfoo –T link:lib foo.m bar.m
```

输出文件如表 3-9 中所示。

表 3-9 生成共享库时的输出文件

文　件	描　述
libfoo.c	库打包器 C 源文件，包含库的输出函数。输出函数表示两个 M 函数（foo.m 和 bar.m）、C 接口和库的初始化代码

续表

文　件	描　述
libfoo.h	库打包器头文件
libfoo_mcc_component_data.c	C 源文件，包含 MCR 初始化和使用库所必需的数据，包括路径信息、密匙和其他初始化信息
libfoo.export	mbuild 链接库时要用到的输出文件
libfoo.ctf	CTF 文档。包含组成库的 M 文件的压缩和加密文档。还包含两个主 M 文件调用的其他文件，以及运行时需要的所有其他可执行内容和数据文件
libfoo.dll	共享库二进制文件

3.3.3 应用程序的部署

用 MATLAB 编译器创建组件以后，可以将它分发或部署到其他机器上，使得其他用户可以在不依赖 MATLAB 的情况下使用该组件。进行产品部署，有下面几点要求：

- 根据生成的应用程序的类型将必要的组件打包。
- 将打好包的组件分发给终端用户。
- 终端用户把它们安装到各自的系统上。在安装处理阶段，终端用户需要在目标机器上运行一次 MCRInstaller 程序。目标机器指的是即将运行应用程序或库的机器。在 Windows 系统中，MCRInstaller 是一个自解压可执行程序，它可安装运行应用程序所必需的组件。

1．跨平台问题

因为二进制格式在不同平台上是不一样的，MATLAB 编译器生成的组件不能从一个平台移植到另一个平台。可以将 MATLAB 编译器生成的应用程序分发到任何与编译机器具有相同操作系统的机器。例如，如果试图将应用程序分发到 Windows 系统的机器，必须用 MATLAB 编译器的 Windows 版本在 Windows 系统的机器上生成应用程序。

将应用程序部署到安装有不同操作系统的机器上，需要重新进行编译。必须在所需要的目标平台上重新编译应用程序。例如，如果希望把前面在 Windows 系统的机器上开发的应用程序部署到 Linux 系统的机器上，必须在 Linux 机器上使用 MATLAB 编译器并完全重新编译应用程序。然后，在两个平台上都要有合法的 MATLAB 编译器许可证。

2．在不执行组件的情况下提取 CTF 文档

CTF 文档包含一些可执行的内容，如 M 文件和 MEX 文件等。执行这些内容以前，需要将它们从文档中提取出来。第一次执行基于 MATLAB 编译器的组件时，CTF 文档会自动展开。这个基于 MATLAB 编译器的组件可以是基于 MATLAB 编译器的独立应用程序或调用基于 MATLAB 编译器的共享库或 COM 组件的应用程序。

使用 extractCTF.exe 独立工具，可以在不运行应用程序的条件下展开文档。该工具位于 <matlabroot>/toolbox/comiler/deploy/<ARCH> 目录下，其中<ARCH>在 Windows 系统中为 win32；在 Linux 系统中为 glnx86。该工具的输入为 CTF 文档的名称，把文档展开到当前工作目录。例如，下面的命令将 hello.ctf 展开到当前工作目录。

```
extractCTF hello.ctf
```

文档内容展开到一个名为 hello_mcr 的目录中。通常，包含展开后的文档的目录名为 <componentname>_mcr，其中 componentname 为 CTF 文档的名称去掉扩展名。

注意：从任何目录运行 extractCTF，必须将<matlabroot>/toolbox/compiler/deploy/<ARCH>

添加到 PATH 环境变量。

3. 交互处理编译路径

MATLAB 编译器使用依赖分析函数（depfun）来确定包含在 CTF 文档中的必需的文件列表。有些情况下，该处理包含过多的文件数目。例如，如果编译过程中包含了 MATLAB 面向对象编程的类，并且编译时不能确定使用了哪个重载方法，就可能返回所有重载方法。依赖分析是一种交互式的处理方法，它还处理每次通过时的 include/exclude 信息。所以，当 CTF 文档很大时，这种方法处理小程序也会耗费很长时间。

减少文件数目最有效的方法是限制编译时 depfun 函数使用的 MATLAB 路径。编译器包含了一些特性，使得可以对路径进行操作。目前有 3 种方法来交互处理编译路径：

- MATLAB 的 addpath 和 rmpath 函数；
- 在 mcc 命令行传递–I <directory>；
- 在 mcc 命令行传递–N 和–p 目录。

（1）使用 MATLAB 的 addpath 和 rmpath 函数

如果从 MATLAB 命令提示中运行编译器，可以在编译前用 addpath 和 rmpath 命令修改 MATLAB 路径。这样做有 2 个缺点：

- 路径只为当前 MATLAB 进程修改；
- 如果编译器在 MATLAB 以外运行，则本方法无效，除非在 MATLAB 中运行了 savepath。

（2）在命令行传递–I <directory>选项

可以用–I 选项将一个目录添加到当前编译过程使用的路径前面。这一点在编译目前不在 MATLAB 路径上的目录中的文件时有用。

（3）在命令行传递–N 和–p <directory>选项

现在有 2 个新的编译器选项，它们提供了更详细的路径操作方法。对于给定的编译过程，这个新特性就像是一个用于 MATLAB 路径的过滤器。第一个选项是–N，在 mcc 命令行传递–N，可以有效清除下面核心目录以外的所有目录的路径。

<matlabroot>/toolbox/matlab

<matlabroot>/toolbox/local

<matlabroot>/toolbox/compiler

它还保留上面列表中编译时出现在 MATLAB 路径中的目录的所有子目录。在命令行包含–N 选项，还使得可以从原始路径中替换目录。另外，如果包含的目录也出现在原始路径上，则包含该目录的所有子目录。

使用–p 选项，在顺序敏感上下文中给编译路径添加一个目录。所谓顺序敏感，指的是与 MATLAB 路径中的顺序相同。语法为：

–p <directory>

这里，<directory>是要包含的目录。如果<directory>不是绝对路径，则假设它位于当前工作目录下。包含这些目录的规则为：

- 如果用–p 选项包含的目录位于原始 MATLAB 路径中，则该目录及其出现在原始路径中的所有子目录按照次序敏感上下文添加到编译路径中。
- 如果用–p 选项包含的目录不在原始 MATLAB 路径中，则该目录不参与编译（可以用–I 选项添加它）。
- 如果用–I 选项添加了某路径，传递了–N 选项，并且该路径已经位于 MATLAB 路径中，

则按照次序敏感上下文添加该路径。否则将路径添加到路径的起始处。

3.3.4 使用 MCR

终端用户在他们的机器上运行 MATLAB 编译器生成的组件以前，如果以前没有安装 MCR，需要安装 MCR。只需要在部署机器上安装一次。

对于 Windows 用户，只需要简单地用 MCRInstaller 工具（MCRInstaller.exe）安装 MCR，按照下面的步骤进行：

（1）找到 <matlabroot>\toolbox\compiler\deploy\win32 目录下的 MCRInstaller 工具 MCRInstaller.exe，并将它复制到新目录下。运行它，解压出几个文件。

（2）运行解压出的 Setup.exe 程序，显示“MATLAB Component Runtime”启动窗口，如图 3-2 所示。单击“Next”按钮，开始安装。



图 3-2 “MATLAB Component Runtime”窗口

（3）启动安装导航，单击“Next”按钮，显示“Select Installation Folder”对话框，如图 3-3 所示。

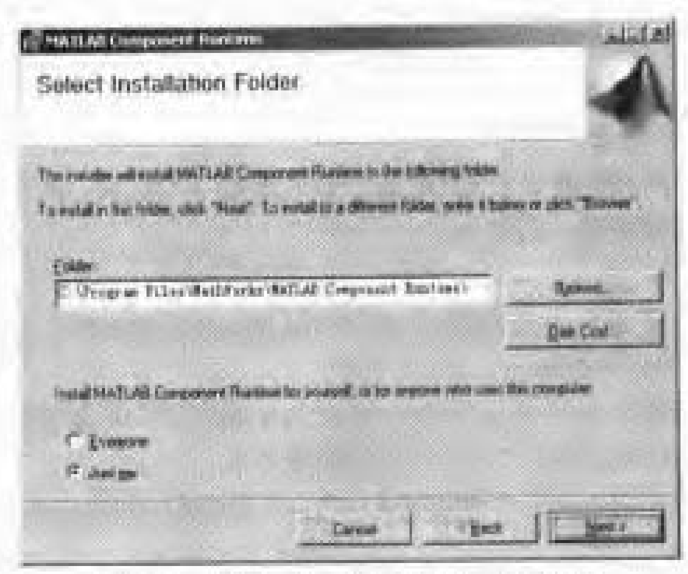


图 3-3 “Select Installation Folder”对话框

（4）在该对话框中选择安装 MCR 的位置。使用该对话框还可以察看系统上可用的和必要的磁盘空间。设置选项，然后单击“Next”按钮继续。

（5）单击“Next”按钮，确认选择。开始安装，根据安装的文件的多少，安装过程需要一定的时间。

（6）安装完毕以后，在“Installation Completed”对话框中单击“Close”按钮。

3.4 使用 mcc

mcc 是调用 MATLAB 编译器的命令，进行 MATLAB 程序编译，需要熟练掌握该命令的用法。

3.4.1 命令概况

mcc 是调用 MATLAB 编译器的 MATLAB 命令，可以从 MATLAB 命令行或 DOS 命令行发出 mcc 命令。

1. 编译器选项

可以给 mcc 指定一个或多个 MATLAB 编译器选项。大部分选项的名称只有一个字母，可以单独列出各选项，如

```
mcc –m –g myfun
```

宏是 MATLAB 提供的编译器选项，它们简化了更通用的编译任务。不必手工组合多个选项来完成特定类型的编译任务，可以使用简单的宏选项。总是可以通过使用单个选项定制编译处理来满足特定的需求。

2. 组合选项

可以把不带参数的选项组合到一起，例如，

```
mcc –mg myfun
```

带参数的选项不能组合，除非将选项和它的参数放在列表最后。例如，下面的格式是合法的。

```
mcc –v –W main –T link:exe myfun      %选项单独列出
mcc –vW main –T link:exe myfun        %合并选项
```

下面的格式不合法

```
mcc –Wv main –T link:exe myfun
```

有一个以上的选项带参数时，只能在组合列表中包含这些选项中的一个，并且该选项必须位于最后。可以将多个组合列表放在 mcc 命令行中。

如果在 mcc 命令行中包含任何 C 或 C++文件名，这些文件与编译器生成的任何 C 或 C++文件一起直接传递给 mbuild 命令。

3. 命令行中的冲突选项

如果使用冲突选项，编译器优先考虑最右侧的选项，例如，

```
mcc –m –W noue test.m
```

等价于

```
mcc –W main –T link:exe –W none test.m
```

本例中，有 2 个冲突的–W 选项。编译器优先考虑最右侧的选项，即–W none，编译器不生成打包器。

4．**设置默认选项**

如果有一些命令行选项，你总是希望将它们传递给 mcc，可以通过设置一个 mccstartup 文件来实现。创建一个包含必要命令行选项的文本文件并命名为 mccstartup。将该文件放在下面两个目录的某一个中：

- 当前工作目录。
- 优先目录<system root>\profiles\<user>\application data\mathworks\matlab\R14。

mcc 在这两个目录中按照上面的顺序搜索 mccstartup 文件。如果找到一个 mccstartup 文件，读取它并处理文件中的选项，就像出现在 mcc 命令行一样。mccstartup 文件和–B 选项按照相同的方式进行处理。

3.4.2 使用宏简化编译

MATLAB 编译器通过其详尽的选项集来获取完成工作所必需的工具。如果需要一个简化的方法来进行编译，可以使用一个简单的选项，即宏。使用它可以快速完成基本的编译任务。宏通过将多个选项组合到一起来完成特定类型的编译任务。

表 3-10 显示了完成标准编译的宏方法与多选项等价方法之间的关系。

表 3-10 宏选项

宏 选 项	束 文 件	创建的对象	等 价 选 项	
			打 包 器	输 出
–l	macro_option_l	库	–W lib	–T link:lib
–m	macro_option_m	独立 C 应用	–W main	–T link:exe

使用–m 选项，让编译器生成独立 C 应用程序。–m 选项等价于下面的系列选项

–W main –T link:exe

表 3-10 显示了组成–m 宏的选项，以及它们提供给编译器的选项。

可以通过编辑<matlabroot>\toolbox\compiler\bundles 目录下的 macro_option 文件来改变宏选项的意义。例如，要改变–m 宏，编辑 bundles 目录下的 macro_option_m 文件。

3.4.3 使用路径名

如果在命令行中给 M 文件指定完整的路径名，则 MATLAB 编译器：

（1）把完整名称分成对应的路径名和文件名(<path>和<file>)。

（2）用“–I <path> <file>”替换参数列表中的完整路径名。例如

mcc –m /home/user myfile.m

将被看作

mcc –m –I /home/user myfile.m

在极少数情况下，这种处理会导致混乱。例如，假设有两个不同的 M 文件，名称都是 myfile.m，且位于/home/user/dir1 和/home/user/dir2 目录下。下面的命令

mcc –m –I /home/user/dir1 /home/user/dir2/myfile.m

与下面的命令等价

mcc –m –I /home/user/dir1 –I /home/useer/dir2 myfile.m

因为–I 选项的缘故，编译器编译 dir2 目录中的 myfile.m 文件时也会找到 dir1 目录中的 myfile.m 文件并编译它。如果意识到可能出现这个错误，可以指定–v 选项，然后看编译器解析了哪个 M 文件。在依赖性分析阶段，–v 选项会把完整路径名输出到 M 文件中。

3.4.4 使用束文件

束文件提供了组合编译器选项并在必要时重复调用它们的便捷途径。束文件选项的语法格式为：

–B <filename>[:<a1>,<a2>,…,<an>]

在 mcc 命令行中使用时，–B 选项用指定文件的内容替换整个字符串。该文件应该只包含 mcc 命令行选项和对应参数和/或其他文件名。文件可以包含其他–B 选项。

束文件可以包含接受名称和版本号的编译器选项的替换参数。例如，有一个用于 C 共享库的束文件 csharedlib，由下面的语法组成。

–W lib:%1% –T link:lib

为了调用编译器生成使用本束文件的 C 共享库，可以使用

mcc –B csharedlib:mysharedlib myfile.m myfile2.m

通常，束文件中的每个%n%会被替换为指定给束文件的对应选项。使用%%包含一个%字符。传递太多或太少选项给束文件都是错误的。

表 3-11 显示了可用的束文件。

表 3-11 可用的束文件

束 文 件 名	创建的对象	内 容
ccom	COM 对象	–W com:<component_name>,<coass_name>,<version> –T link:lib
cexcel	Excel COM 对象	–W excel:<component_name>,<coass_name>,<version> –T link:lib –b
cppcom	COM 对象	–B ccom:<component_name>,<coass_name>,<version>
cppexcel	Excel COM 对象	–B cexcel:<component_name>,<coass_name>,<version>
cpplib	C++库	–B csharedlib:<shared_library_name> –T compile:lib
csharedlib	C 共享库	–W lib:<shared_library_name> –T link:lib
macro_option_l	N/A	–W lib –T link:lib
macro_option_m	N/A	–W main –T link:exe

注意：*用 MATLAB 编译器创建 COM 组件，必须安装 MATLAB 的 COM 生成器工具。创建 Microsoft Excel 插件，必须安装 Excel 生成器工具。*

3.4.5 使用打包器文件

打包器文件包含多个打包器函数，通过提供必要的接口，打包器文件可以创建 MATLAB 编译器生成的代码与所支持的可执行程序，如独立应用或库之间的链接。这些接口允许代码在必要的执行环境中进行操作。

为了提供必要的接口，打包器进行打包器指定的初始化和终止；提供函数调用 MCR 的分派指令。

指定打包器类型，使用下面的语法：

–W <type>

1．主文件打包器

–W main 选项生成适合创建独立应用的打包器。这些依据 POSIX 的主打包器从 POSIX 解释器位置接受字符串，返回状态码。它们把这些命令行字符串作为 MATLAB 字符串传递给 M 文件函数。

对于下面的 M 文件 sample.m

```
function y=sample(varargin)
varargin{:}
y=0;
```

可以将 sample.m 编译为 POSIX 主应用程序。如果在 MATLAB 中调用 sample 函数，结果类似下面：

```
sample hello world
ans=
hello
ans=
world
ans=
    0
```

如果编译 sample.m 并且从 DOS 环境调用它，结果类似下面

```
C:\> sample hello world
ans=
hello
ans=
world
C:\>
```

2．C 库打包器

–l 选项或它的等价语法–W lib :libname 生成 C 库打包器文件。该选项用任何 M 文件集合生成共享库。对于每个编译后的 M 文件，该选项生成包含 C 函数声明的头文件，输出列表包含 C 共享库输出的符号集。

注意：从更大的应用程序中调用任何编译器生成的代码时，必须生成库打包器文件。

3．C++库打包器

–W cpplib:libname 选项生成 C++库打包器文件。该选项允许将任何 M 文件集包括到库中。生成的头文件包括所有编译过的 M 函数的入口点。

4．COM 组件打包器

使用 COM 组件打包器文件可以用 MATLAB M 文件创建 COM 组件。生成 COM 打包器的选项为

–W com:<component_name>[,<coass_name>[,<major>.<minor>]]

–W excel:<component_name>[,<coass_name>[,<major>.<minor>]]

COM 打包器选项创建生成 C 或 C++打包器时创建的文件的超集。

3.4.6 使用注记

在独立 C 和 C++模式中，注记

%#function <function_name-list>

通知 MATLAB 编译器，编译过程中应该包含指定的函数，而不管编译器的依赖分析是否发现了它。没有这个注记，编译器的依赖分析将找不到，并且不能编译应用程序用到的所有 M 文件。

不能用%#function 注记引用 M 代码中没有的函数。

不管在哪里使用 feval 语句，在代码中使用%#function 都是好的编码技巧。下面的例子演示如何使用这个技巧来帮助编译器在编译时找到合适的文件，不需要在命令行包含所有文件。

3.4.7 脚本文件

MATLAB 编译器不能编译脚本式 M 文件，但是可以编译调用脚本的函数式 M 文件。不可以在 mcc 命令行显式指定脚本式 M 文件，但可以指定包含脚本本身的函数式 M 文件。

将脚本转换为函数通常很简单，只需在 M 文件的顶行添加函数行就行了。

例如，考虑下面的脚本式 M 文件 houdini.m。

```
m=magic(4);
t=m.^3;
disp(t);
```

运行本脚本式 M 文件，可以在 MATLAB 工作空间中创建变量 m 和 t。

MATLAB 编译器不能编译 houdini.m，因为它是一个脚本。将该脚本转换为函数，只需要添加一个函数行，即

```
function houdini(sz)
m=magic(4);
t=m.^3;
disp(t);
```

现在 MATLAB 编译器可以编译 houdini.m 了。但是，因为现在 houdini 是一个函数了，运行 houdini.m 不再在 MATLAB 工作空间中创建变量 m 和 t。如果从 MATLAB 工作空间中获得变量 m 和 t 很重要，可以将函数形式改为：

```
function [m,t]=Houdini(sz)
```

现在，调用该函数时会返回 m 和 t 的值。

编译后的应用程序由两层 M 文件组成。顶层为接口层，由那些直接可以从 C 或 C++获取的函数组成。

在独立应用中，接口层只包括主 M 文件。在库中，接口层由 mcc 命令行指定的 M 文件组成。

M 文件的第 2 层包括那些被顶层函数调用的 M 文件。可以在第 2 层包括脚本，但不能在顶层包括。

3.5 独立应用程序

假设要创建一个计算大型魔方矩阵的秩的应用程序，方法之一是用 C 或 C++编写整个代

码，这需要自己编写与魔方矩阵、秩和奇异值等有关的函数。一个更简单的办法是将它编写成一个或多个 M 文件，然后利用 MATLAB 及其工具的威力来实现。

可以创建使用 MATLAB 数学函数的应用程序，但不要求终端用户拥有自己的 MATLAB。独立应用程序是将 MATLAB 的功能打包并分发给用户的一种比较方便的途径。

独立 C 应用程序的源代码可以全部为 M 文件，也可以是 M 文件、MEX 文件或 C/C++源代码文件的组合。

MATLAB 编译器将 M 文件生成为 C 源代码函数，使得可以从 MATLAB 交互环境以外调用 M 文件。编译这些 C 源代码以后，生成的对象文件与运行时库相连接。生成 C++独立应用程序进行相似的处理。

可以从编译器生成的独立应用程序中调用 MEX 文件。然后该 MEX 文件被载入，并被独立代码调用。

3.5.1 C 独立应用程序

本节提供一个实例，演示从应用程序编译到分发的全过程。本例将 M 文件 magicsquare.m 创建为独立 C 应用程序 magicsquare。

1．编译应用程序

（1）将文件 magicsquare.m 从<matlabroot>\extern\examples\compiler 复制到工作目录。

（2）编译该 M 代码，在命令行输入

```
mcc –mv magicsquare.m
```

–m 选项让 MATLAB 编译器(mcc)生成一个 C 独立应用。–v 选项显示编译步骤并帮助识别其他有用信息。例如使用了哪个第三方编译器，引用了哪些环境变量等。

本命令创建名为 magicsquare 的独立应用程序及其他文件。Windows 平台在文件名后面添加.exe 扩展名。

2．测试应用程序

按照下面的步骤，在开发独立应用程序的机器上测试应用程序。

（1）更新路径。在工作路径中添加下面的目录。

```
<matlabroot>\bin\win32
```

（2）运行独立应用程序

```
magicsquare.exe 4
ans=
    16   2    3    13
    5    11   10   8
    9    7    6    12
    4    14   15   1
```

3．部署应用程序

可以将 MATLAB 编译器生成的独立应用程序分发到任何具有相同操作系统的机器上。例如，如果是在使用 Windows 系统的机器上生成的独立程序，就只能在使用 Windows 系统的机器上部署该应用程序。如果需要跨平台使用，则需要相关平台的 MATLAB 和 MATLAB 编译器许可。

在 Windows 系统下，需要搜集和打包表 3-12 中的文件并将它们分发到进行部署的机器上。

表 3-12　需要搜集和打包的文件

组　件	描　述
MCRInstaller.exe	自解压 MATLAB 组件运行时库工具；与平台有关的文件，必须与终端用户的平台相对应；该文件位于<matlabroot>\toolbox\compiler\deploy\win32 目录下
magicsquare.ctf	组件技术文件，为 CTF 文件；与平台有关
magicsquare	应用程序；在 Windows 系统下为 magicsquare.exe

4. 运行应用程序

按照下面的步骤，在终端用户的机器上运行独立应用程序。

（1）通过运行 MCR 安装器来安装 MCR。

（2）将可执行文件和 CTF 文档复制到应用程序的根目录中，例如 C:\approot。

（3）在系统路径中添加下面的目录。

<mcr_root>\runtime\win32

（4）运行 magicsquare 独立应用程序，并提供一个表示魔方大小的数字。

```
magicsquare 4
ans=
    16   2    3    13
    5    11   10   8
    9    7    6    12
    4    14   15   1
```

3.5.2 源代码只包括 M 文件

创建独立应用程序的方法之一是将所有源代码编写在一个或多个 M 文件或 MEX 文件中，就象前面的魔方实例一样。用 M 文件编写应用程序的源代码，可以利用 MATLAB 交互开发环境。只要程序的 M 文件运行良好，就可以进行编译并生成独立应用程序。

考虑一个简单的应用程序，其源代码包括两个 M 文件，即 mrank.m 和 main.m。本例根据 M 文件生成 C 代码。

mrank.m 返回一个整数矢量 r。r 的每个元素表示魔方矩阵的秩。例如，函数完成以后，r(3)包括 3×3 魔方矩阵的秩。

```
function r=mrank(n)
r=zeros(n,1);
for =1:n
    r(k)=rank(magic(k));
end
```

本例中，行 r=zeros(n,1)通过预分配内存空间来改善编译器的运行效率。

main.m 包含一个主函数，它调用 mrank 函数并输出结果。

```
function main
r=mrank(5)
```

将这些文件编译为可以生成独立应用程序的代码，调用 MATLAB 编译器，如下所示：

```
mcc –m main mrank
```

–m 选项使 MATLAB 编译器生成可以创建独立应用程序的 C 代码。例如，MATLAB 编译器生成源代码文件 main.c,main_main.c 和 mrank.c。main_main.c 包含一个名为 main 的 C 函数；

main.c 和 mrank.c 包含名为 mlfMain 和 mlfMrank 的 C 函数。

为了生成可执行应用程序，可以用 mbuild 编译和连接这些文件。或者，可以使用下面的命令自动化整个过程。

```
mcc –m main mrank
```

如果需要将其他代码组合到应用程序中去，或者想生成编译应用程序的生成文件(makefile)，可以使用下面的命令。

```
mcc –mc main mrank
```

3.5.3 源代码包含 M 文件和 C/C++文件

创建独立应用程序的另--种方法是将一部分代码用一个或多个函数式 M 文件编写，另一部分代码直接用 C 或 C++编写。使用这种编写方式，必须懂得调用 MATLAB 编译器生成的外部 C 或 C++函数，并知道如何使用这些 C 或 C++函数返回的结果。

3.6 库

可以用 MATLAB 编译器将 MATLAB 算法打包成 C 或 C++共享库，然后在用 C 或 C++编程时调用这些共享库中的函数，就象从 MATLAB 命令行调用这些函数一样。

3.6.1 C 共享库

可以用 MATLAB 编译器生成 C 或 C++共享库。很多用于创建独立应用程序的 mcc 选项也可以用于创建 C 和 C++共享库。

1. C 共享库打包器

使用 C 库打包器选项可以用任何 M 文件集创建共享库。MATLAB 编译器生成一个打包器文件、一个头文件和一个输出列表。头文件包含所有编译后的 M 函数的所有入口点。输出列表包括从 C 共享库输出的符号集。

2. C 共享库示例

下面的例子用几个 M 文件创建一个 C 共享库，还包括一个调用共享库的独立驱动程序。

（1）生成共享库

把下面的文件从<matlabroot>\extern\examples\compiler 目录下复制到你的工作目录。

<matlabroot>\extern\examples\compiler\addmatrix.m

<matlabroot>\extern\examples\compiler\multiplymatrix.m

<matlabroot>\extern\examples\compiler\eigmatrix.m

<matlabroot>\extern\examples\compiler\matrixdriver.c

注意：matrixdriver.c 包含独立应用程序的主函数。

使用下面的命令创建共享库。

```
mcc –B csharedlib:libmatrix addmatrix.m multiplymatrix.m eigmatrix.m –v
```

–B csharedlib 选项是一个束选项，它扩展为

```
–W lib:<libname> –T link:lib
```

–W lib:<libname>选项让 MATLAB 编译器生成一个函数打包器，名称为 libname。–T

link:lib 选项指定目标输出为共享库。

（2）编写驱动程序

调用 MATLAB 编译器生成的共享库的所有程序都有大致相同的结构：

- 声明变量，处理和验证输入参数。
- 调用 mclInitializeApplication 函数，测试是否成功。该函数设置全局 MCR 状态并使得可以创建 MCR 的实例。
- 每个库调用一次<libraryname>Initialize，创建库所需要的 MCR 实例。
- 调用库中的函数并处理结果，这是程序的主体。
- 每个库调用一次<libraryname>Terminate，销毁相关的 MCR 实例。
- 调用 mclTerminateApplication 函数，释放与全局 MCR 状态有关的资源。
- 清除变量，关闭文件等，退出。

本例使用 matrixdriver.c 作为驱动程序。

（3）编译驱动程序

编译驱动代码 matrixdriver.c，使用 C/C++编译器。执行下面的 mbuild 命令，用 C/C++编译器编译代码。

```
mbuild matrixdriver.c libmatrix.lib
```

注意：本命令假设共享库和第 2 步创建的对应的头文件位于当前工作目录中。如果不在，则指定完整路径给 libmatrix.lib。

这里生成一个独立应用程序 matrixdriver.exe。

输出的 mlf 函数的定名规则一般为：

- 对于没有返回值的 M 函数

```
void mlf<function-name>(<list_of_input_variables>);
```

- 对于至少有一个返回值的 M 函数

```
void mlf<function-name>(int number_of_return_values),
<list_of_pointer_to_return_variables>,
<list_of_input_variables>;
```

（4）测试驱动程序

按照下面的步骤在开发机器上测试独立驱动程序和共享库。

- 要运行独立应用程序，添加包含共享库的目录。共享库在前面已经创建好了。
- 更新动态库路径。在动态库路径中添加下面的目录。

```
<matlabroot>\bin\win32
```

- 键入应用程序的名称，运行驱动程序。

```
matrixdriver.exe
```

结果显示为：

```
The value of the added matrix is:
2.00        4.00        6.00
8.00        10.00       12.00
14.00       16.00       18.00

The value of the multiplied matrix is:
30.00       36.00       42.00
```

66.00 81.00 96.00
102.00 126.00 150.00

The eigenvalue of the first matrix is:
16.12 –1.12 –0.00

（5）部署调用共享库的独立应用程序

部署调用共享库的独立应用程序，需要搜集和打包表 3-13 中的文件并把它们分发到部署机器上。

表 3-13 部署调用共享库的独立应用程序需要搜集和打包的文件

组　件	描　述
MCRInstaller.exe	自解压 MATLAB 组件运行时库工具；创建该文件的平台必须与终端用户的平台一致
matrixdriver.ctf	组件技术文件，与平台有关
matrixdriver.exe	应用程序
libmatrix.dll	共享库

（6）部署共享库，用于其他工程

使共享库可以用于外部程序，需要分发表 3-14 中的组件。

表 3-14 将共享库用于外部程序需要分发的组件

组　件	描　述
MCRInstaller.exe	自解压 MATLAB 组件运行时库工具；创建该文件的平台必须与终端用户的平台一致
libmatrix.ctf	组件技术文件，与平台有关
libmatrix.h	库的头文件
libmatrix.dll	共享库

3．部署共享库

运行时，有一个 MCR 实例与每个单独的共享库相连。所以，如果应用程序连接了两个编译器生成的共享库，则运行时会创建两个 MCR 实例。

可以用 MCR 选项控制每个 MCR 实例的行为。MCR 选项可以分为全局的和局部的。全局 MCR 选项程序中的每个 MCR 实例都可以识别，局部 MCR 选项则根据 MCR 实例的情况有所区别。

使用共享库，必须使用下面的函数：

- mclInitializeApplication
- mclTerminateApplication

使用 mclInitializeApplication 函数，可以设置全局 MCR 选项，它们对所有 MCR 实例同样适用。必须在创建 MCR 实例以前设置这些选项。

这些函数是必要的，因为有些选项，如是否启动 Java、MCR 的位置、是否使用 MATLAB 的 JIT 功能等，都在创建第一个 MCR 实例时进行设置，而且以后生成的 MCR 实例不能改变它。

注意：*使用驱动程序以前必须调用一次 mclInitializeApplication 函数，且必须在调用其他任何 MATLAB 函数以前调用它。*

函数调用形式为：

```
bool mclInitializeApplication(const char **options,int count);
```

```
bool mclTerminateApplication(void);
```

mclInitializeApplication 函数有两个输入参数，一个是用户可以设置的选项字符串数组，另一个是选项的个数。如果成功，返回 true，否则返回 false。

mclTerminateApplication 函数没有输入参数，只能在所有 MCR 实例被销毁以后调用。成功时返回 true，否则返回 false。

注意：调用 mclTerminateApplication 函数以后，不能再调用 mclInitializeApplication 函数。调用 mclTerminateApplication 函数以后不能再调用任何 MATLAB 函数。

下面的函数显示了这些函数的典型应用。

```
int main(){

    mxArray *in1, *in2; /* Define input parameters */
    mxArray *out = NULL;/* and output parameters to be passed to
              the library functions */

    double data[] = {1,2,3,4,5,6,7,8,9};

    /* Call the library intialization routine and make sure that
    the library was initialized properly */
    mclInitializeApplication(NULL,0);
    if (!libmatrixInitialize()){
        fprintf(stderr,"could not initialize the library
          properly\n");
        return -1;
    }

    /* Create the input data */
    in1 = mxCreateDoubleMatrix(3,3,mxREAL);
    in2 = mxCreateDoubleMatrix(3,3,mxREAL);
    memcpy(mxGetPr(in1), data, 9*sizeof(double));
    memcpy(mxGetPr(in2), data, 9*sizeof(double));

    /* Call the library function */
    mlfAddmatrix(1, &out, in1, in2);
    /* Display the return value of the library function */
    printf("The value of added matrix is:\n");
    display(out);
    /* Destroy the return value since this variable will be rensed
    in the next function call. Since we are going to reuse the
    variable, we have to set it to NULL. Refer to MATLAB
    Compiler documentation for more information on this. */
    mxDestroyArray(out); out=0;
    mlfMultiplymatrix(1, &out, in1, in2);
    printf("The value of the multiplied matrix is:\n");
    display(out);
    mxDestroyArray(out); out=0;
    mlfEigmatrix(1, &out, in1);
    printf("The Eigen value of the first matrix is:\n");
    display(out);
```

```
        mxDestroyArray(out); out=0;

        /* Call the library termination routine */
        libmatrixTerminate();

        /* Free the memory created */
        mxDestroyArray(in1); in1=0;
        mxDestroyArray(in2); in2 = 0;
        mclTerminateApplication();
        return 1;
    }
```

注意：对于每个程序，mclInitializeApplication 函数只能被调用一次。调用第 2 次将导致错误，并导致函数返回 false。该函数必须在调用任何 C MX 函数或 MAT 文件 API 函数以前被调用。

必须按照下面的步骤在应用程序中使用 MATLAB 编译器生成的共享库。

（1）在应用程序中包含每个库的头文件， MATLAB 生成的每个共享库都有一个相关的头文件，名为<lib-name>.h，其中<lib-name>为库名。编译库时它在命令行进行传递。

（2）通过调用 mclInitializeApplication 函数初始化 MATLAB 库。每个程序必须调用一次该函数，并且必须在调用其他任何 MATLAB API 函数以前调用它。

（3）对于应用程序包含的每个 MATLAB 共享库，必须调用库的初始化函数。该函数进行几个方面的初始化，例如解压 CTF 文档，并生成一个带有必要信息的 MCR 实例来执行文档中的代码。库初始化函数将被命名为<lib-name>Initialize()，其中<lib-name>为库名。本函数还返回一个 Boolean 型状态代码。返回值为 true 时表示初始化成功，为 false 时表示失败。

（4）必要时调用每个库的输出函数，用 C MX API 函数处理这些函数的输入、输出参数。

（5）应用程序不再需要给定的库时，调用库的终止函数。该函数释放与 MCR 实例有关的资源。库终止函数的名称为<lib-name>Terminate()，其中<lib-name>为库名。一旦库被终止，库的输出函数在应用程序中将不再调用。

（6）程序不再需要调用 MATLAB 编译器生成的任何库时，调用 mclTerminateApplication API 函数。本函数释放 MCR 使用的应用程序级别的资源。一旦调用了该函数，就不能再调用共享库中的任何函数。

3.6.2 C++共享库

使用 C++库打包器选项，可以用任何 M 文件集创建一个共享库。MATLAB 编译器生成一个打包器文件和一个头文件。头文件包含所有编译后的 M 函数的所有入口点。

注意：即使不是在生成共享库，将编译器生成的任何代码包含到更大的应用程序中时也必须使用–W lib 或–W cpplib 选项。

除了使用 cpplib 打包器外，创建 C++共享库的步骤与创建 C 共享库的相同。

```
mcc W cpplib:libmatrix T link:lib addmatrix.m multiplymatrix.m eigmatrix.m –v
```

W cpplib:<libname>选项让 MATLAB 编译器为共享库生成一个名为<libname>的函数打包器。T link:lib 选项指定目标输出类型为共享库。

下面的例子用 C++重写前面的 C 共享库，并按照下面的步骤进行。

（1）编写驱动程序

下面编写 matrixdriver 程序的 C++版本 matrixdriver.cpp。

```
/*==============================================================
 *
 * MATRIXDRIVER.CPP
 * Sample driver code that calls a C++ shared library created using
 * the MATLAB Compiler. Refer to the MATLAB Compiler documentation
 * for more information on this
 *
 * This is the wrapper CPP code to call a shared library created
 * using the MATLAB Compiler.
 *
 * Copyright 1984-2004 The MathWorks, Inc.
 *
 *==============================================================*/

// Include the library specific header file as generated by the
// MATLAB Compiler
#include "libmatrix.h"

int main(){


    // Call application and library initialization. Perform this
    // initialization before calling any API functions or
    // Compiler-generated libraries.
    if (!mclInitializeApplication(NULL,0) ||
    !libmatrixInitialize())
    {
        std::cerr << "could not initialize the library properly"
                  << std::endl;
        return -1;
    }

    try
    {
        // Create input data
        double data[] = {1,2,3,4,5,6,7,8,9};
        mwArray in1(3, 3, mxDOUBLE_CLASS, mxREAL);
        mwArray in2(3, 3, mxDOUBLE_CLASS, mxREAL);
        in1.SetData(data, 9);
        in2.SetData(data, 9);

        // Create output array
        mwArray out;

        // Call the library function
        addmatrix(1, out, in1, in2);
```

```
            // Display the return value of the library function
            std::cout << "The value of added matrix is:" << std::endl;
            std::cout << out << std::endl;

            multiplymatrix(1, out, in1, in2);
        std::cout << "The value of the multiplied matrix is:"
            << std::endl;
            std::cout << out << std::endl;

            eigmatrix(1, out, in1);
            std::cout << "The eigenvalues of the first matrix are:"
                        << std::endl;
            std::cout << out << std::endl;
        }
        catch (const mwException& e)
        {
            std::cerr << e.what() << std::endl;
            return -1;
        }
        catch (...)
        {
            std::cerr << "Unexpected error thrown" << std::endl;
            return -1;
        }

        // Call the application and library termination routine
        libmatrixTerminate();
        mclTerminateApplication();

        return 0;
    }
```

（2）编译驱动程序

用 C++编译器编译 matrixdriver.cpp 驱动程序代码。执行下面的 mbuild 命令，使用 C++编译器编译代码。

```
mbuild matrixdriver.cpp libmatrix.lib
```

（3）把 C++库合并到应用程序中

将 C++共享库合并到应用程序中，与使用 C 共享库的步骤基本相同。但有两点主要的差异：

- 接口函数使用 mwArray 类型而不是 mxArray 类型传递参数。
- 使用 C++容错结构给调用函数报告错误。所以，所有调用必须包括在 try-catch 结构中。

（4）输出函数的形式

C++共享库给每个 M 函数生成两套接口。第一套是输出接口，与 C 共享库中生成的 mlx 名称相同。第二套接口为 C++函数接口。输出 C++函数的一般形式为：

- 对于没有返回值的 M 函数

```
void <function-name>(<list_of_input_variables>);
```

- 对于至少有一个返回值的 M 函数

```
void <function-name>(int number_of_return_values,
        <list_of_return_variables>,<list_of_input_variables>);
```

其中，<list_of_input_variables>表示逗号间隔的 const mwArray&类型的列表，<list_of_return_variables>表示逗号间隔的 mwArray&类型的列表。例如，在 libmatrix 库中，addmatrix M 函数的 C++接口生成为：

```
void addmatrix(int nargout,mwArray& a,const mwArray& a1,
       const mwArray& a2);
```

（5）错误控制

C++接口函数在执行过程中通过抛出一个 C++出错消息来控制错误。使用 mwException 类来达到这个目的。应用程序可以通过捕捉 mwException 并搜索 what()方法来获得出错消息。为了在调用 C++接口函数时能正确控制错误，将每个调用放到 try-catch 块中，即

```
try
{
    (调用函数)
}
catch(const mwException& e)
{
    (控制错误)
}
```

matrixdriver.cpp 应用程序演示了调用 C++接口函数时控制错误的典型方式。

3.6.3 MATLAB 编译器生成的接口函数

MATLAB 编译器生成的共享库至少包含 7 个接口函数。有 3 个函数管理库的初始化和终止，1 个打印输出信息，1 个打印错误信息，2 个负责将每个 M 文件编译到库中。

要生成本节介绍的函数，首先把 sierpinski.m 和 triangle.c 从<matlabroot>\extern\examples\compiler 目录下复制到工作目录，然后执行合适的编译器命令。

对于 C 应用程序，命令为：

```
mcc –W lib:libtriangle –T link:lib sierpinski.m
mbuild triangle.c libtriangle.lib
```

对于 C++应用程序，命令为：

```
mcc –W cpplib:libtriangle –T link:lib sierpinski.m
mbuild triangle.cpp libtriangle.lib
```

这些命令创建一个名为 triangle 的主程序和一个名为 libtriangle 的共享库。该库输出一个使用简单迭代算法的函数来生成不规则碎片形状，这就是所谓的 Sierpinski 三角形。triangle.c 或 triangle.cpp 的主程序有一个可选的数值参数，如果它存在，会指定用于生成不规则碎片的点数。例如，triangle 8000 生成一个有 8000 个点的金刚石图案，如图 3-4 所示。

本例中，MATLAB 编译器将生成的所有函数放在文件 libtriangle.c 或 libtriangle.cpp 中

1．调用共享库的程序的结构

调用 MATLAB 编译器生成的共享库的所有程序具有大致相同的结构，包括：

- 声明参数和处理输入参数；
- 调用 tuclInitializeApplication 函数并测试是否成功；

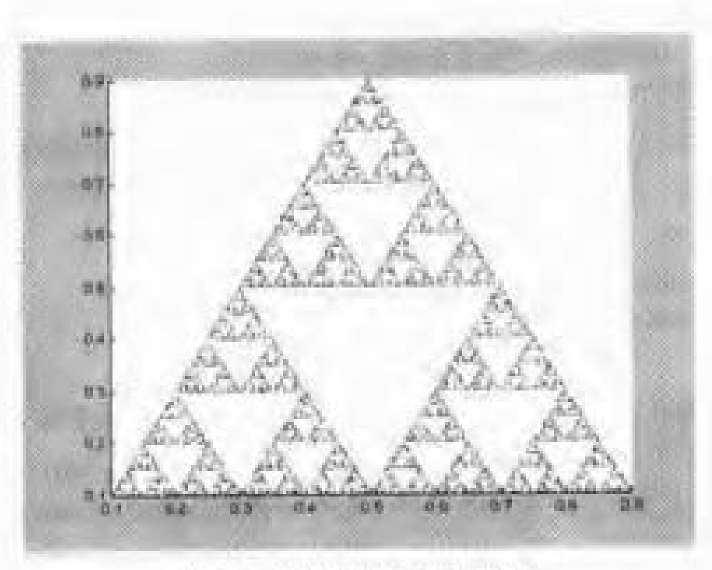


图 3-4　递归生成的金刚石图案

- 每个库调用一次<libraryname>Iniatialize，创建库所需要的 MCR 实例；
- 调用库中的函数并处理结果；
- 每个库调用一次<libraryname>Terminate，销毁相关的 MCR 实例；
- 清除变量，关闭文件并退出。

2．库的初始化和终止函数

库的初始化和终止函数分别创建和销毁共享库所需要的 MCR 实例。必须在调用共享库中的其他函数前调用初始化函数，并且在完成对共享库的调用以后（或为了防止内存泄露）应该调用终止函数。

有两种形式的初始化函数，一个终止函数。两种初始化函数中较简单的一种没有输入参数，大部分情况下会调用这个函数。本例中，初始化函数的这种形式称为 libtriangleInitialize，有：

```
bool libtriangleInitialize(void)
```

该函数用默认的打印和错误控制和编译处理时生成的其他信息创建一个 MCR 实例。但是，如果试图对打印输出和错误控制的方式进行更多的控制，就需要调用该函数的第 2 种形式，它有 2 个参数。形式为：

```
bool libtriangleInitializeWithHandlers(
    mclOutputHandlerFcn error_handler,
    mclOutputHandlerFcn print_handler)
```

通过调用这个函数，可以提供自己的打印和错误控制函数，MCR 会调用它们。这些函数具有相同的形式。通过覆盖默认形式，可以控制输出如何显示，以及是否放到日志文件中等。

在 Windows 平台上，编译器另外生成一个初始化函数，即标准 Microsoft DLL 初始化函数 DllMain。

```
BOOL WINAPI DllMain(HINSTANCE hInstance,DWORD dwReason,void *pv)
```

DllMain 函数提供很重要的服务，它可以确定共享库保存在磁盘的哪个目录下，该信息用于查找 CTF 文档，没有该文档，应用程序将不能运行。如果修改 DllMain 函数，要确保保留它的这部分功能。

库终止函数比较简单，形式为：

```
void libtriangleTerminate(void)
```

在调用 mclTerminateApplication 函数以前调用该函数。

3. 打印和错误控制函数

默认时，MATLAB 编译器生成的应用程序和共享库将打印的输出传送给标准输出，将出错消息传送给标准错误。编译器生成默认的打印控制函数和默认的错误控制函数。如果希望改变本行为，必须编写自己的错误和打印控制函数并将它们传送给合适的初始化函数。

默认的打印控制函数具有下面的形式：

```
static int mclDefaultPrintHandler(const char *s)
```

默认的错误控制函数具有与打印控制函数相同的形式：

```
static int mclDefaultErrorHandler(const char *s)
```

4. 从 M 文件生成的函数

对于 MATLAB 编译器命令行上指定的每个 M 文件，编译器都会生成两个函数，即 mlx 函数和 mlf 函数。这两个函数具有相同的作用，即调用 M 文件函数；具有不同的名称，表示不同的接口。每个函数的名称根据 M 文件中第 1 个函数的名称确定，每个函数前面添加一个 3 个字母的前缀。

（1）mlx 接口函数

以 mlx 前缀打头的函数与 MATLAB MEX 函数具有相同类型和数目的参数。第 1 个参数 nlhs 为输出参数的个数，第 2 个参数 plhs 为指向一个数组的指针，数组中为函数返回值的个数。第 3 个和第 4 个参数为输入参数的个数和一个包含输入变量的数组。

```
void mlxSierpinski(int nlhs,mxArray *plhs[],int nrhs,mxArray *prhs[])
```

（2）mlf 接口函数

以 mlf 前缀打头的函数希望其输入和输出参数作为单独变量，而不是数组进行传递。如果函数可以生成一个或更多输出，第一个参数为输出的个数。

```
void mlfSierpinski(int nargout,mxArray** x,mxArray** y,
                   MxArray* iterations,mxArray* draw)
```

注意，在这两种情况下，生成的函数都会给它们的返回值分配内存。如果处理完输出变量以后不删除内存，将造成内存泄露。

5. 在 M 函数接口中使用 varargin 和 varargout

如果 M 函数接口使用 varargin 或 varargout，则必须将它们作为单元数组传递。例如，如果有 N 个 varargin，则需要创建一个大小为 1×N 的单元数组。类似地，varargout 也作为一个单元数组返回。varargout 的长度等于返回值的个数减去实际传递的变量数。在 MATLAB 中，表示 varargout 的单元数组必须是最后一个返回变量，表示 varargin 的单元数组必须是最后一个形参。

对于下面的 M 文件接口

```
[a,b,varargout]=myfun(x,y,z,varargin)
```

对应的 C 接口为：

```
void mlfMyfun(int numOfRetVars,mxArray **a,mxArray **b,
      mxArray **varargout,mxArray *x,mxArray *y,
      mxArray *z,mxArray *varargin)
```

本例中，varargout 的元素个数为(numOfRetVars–2)，其中 2 表示返回的 2 个实际变量 a

和 b。varargin 和 varargout 都是单行多列的单元数组。

3.7 COM 和 Excel 组件

将 MATLAB 提供的 COM 生成器和 Excel 生成器与 MATLAB 编译器一起使用，可以创建 COM 对象和 Excel 插件。

3.7.1 生成 COM 组件

用 MATLAB 编译器创建 COM 组件，必须先在系统上安装 COM 生成器。可以用 MATLAB 编译器，根据 MATLAB M 文件创建 COM 对象。M 文件集合会传递到一个单独的 COM 类中。COM 生成器支持一个组件有多个类的情况。

COM 类的接口与 C 共享库导出的一套函数相同，但 MATLAB 编译器对创建 COM 对象时生成的 C 和 C++代码都提供支持。

mcc 命令会自动完成下面的工作：

- 调用微软接口定义语言（MIDL）编译器；
- 调用资源编译器；
- 指定 DEF 文件。

使用 mcc 命令的选项，可以完成 COM 兼容 DLL 的自动注册。

例如，将 plus1.m 编译成 COM 对象，使用下面的语法：

```
mcc -B 'ccom:addin,addin,1.0' plus1.m
```

使用 COM 打包器文件可以根据 MATLAB M 文件创建 COM 组件。生成 COM 打包器的编译器选项为：

```
-W com:<component_name>[,<class_name>[,<major>.<minor>]]
-W excel:<component_name>[,<class_name>[,<major>.<minor>]]
```

COM 打包器选项会创建一些文件的超集，这些文件是在生成 C 或 C++库打包器时创建的。除了 C 或 C++库文件外，COM 打包器会创建表 3-15 中的文件。

表 3-15 COM 打包器创建的文件

文 件	描 述
<component_name>_idl.idl	COM 的接口描述文件
<component_name>_com.hpp	COM 类的 C++头文件
<component_name>_com.cpp	COM 类的 C++源文件
<component_name>_dll.cpp	COM 对象的 DLL 接口
<component_name>_def	COM DLL 的定义文件

如果没有指定<class_name>，则默认时为<component_name>。如果没有指定版本号，则默认时采用最后生成时的版本，或者没有先前版本时指定为 1.0。

COM 打包器选项生成创建一个单独的 COM 对象所必需的所有代码和文件，该对象包含编译器生成的所有接口。打包器选项创建一个与指定的<class_name>名称相同的 COM 类和一个对应的名为<class_name>的接口类。使用主版本号和次版本号来控制所生成的 COM 接口的主版本号和次版本号。

COM 生成器用 C++而不是 C 生成 COM 接口。只是 COM 有这方面的要求，对 MATLAB

编译器来说并没有。COM 生成器自动将生成的所有 C++源文件添加到 mcc 命令行。

如果指定了主版本号和次版本号，编译器会用指定的新版本号替换任何已经存在的类型库。如果没有指定版本号并且有一个已经存在的类型库，编译器会替换当前版本。

3.7.2 生成 Excel 插件

用 MATLAB 编译器创建 Excel 插件，必须在系统上安装 MATLAB 的 Excel 生成器。使用 Excel 生成器，可以根据 MATLAB M 文件自动生成 Visual Basic 模块文件（.bas）和 DLL 插件，可以将它们作为独立的函数导入到 Excel 中。

例如，将 plus1.m 编译为 Excel 插件，使用下面的命令行：

```
mcc –B 'cexcel:addin,addin,1.0' plus1.m
```

Excel 生成器生成的 COM 类与 COM 生成器生成的 COM 类具有相同的结构，并服从相同的标准。

第 4 章　MATLAB 调用动态链接库

共享库是系统上的一个或更多应用程序可以使用的函数的集合。在 Windows 系统中，它被预编译为动态链接库（.dll）文件。运行时，该库载入内存，并且所有应用程序都有权使用它。MATLAB 提供了调用 DLL 的接口，使得可以直接从 MATLAB 调用动态链接库中的函数。

在 MATLAB 中通过命令行接口很容易将 C 程序生成为外部共享库。使用该接口可以将外部库载入 MATLAB 内存空间，然后调用其中定义的任何函数。尽管两种语言环境有区别，但是大多数情况下可以直接将 MATLAB 的数据类型传递给 C 函数，不必进行转换。MATLAB 会帮你进行转换。

该接口还支持包含除 C 外其他语言编写的函数组成的库（如果这些函数有 C 接口）。

4.1　库的载入和卸载

在 MATLAB 有权调用共享库中的外部函数以前，必须首先将该库载入内存。一旦载入，就可以查找库中任何函数的信息并直接从 MATLAB 中调用它们。不再需要库时，就要从内存中卸载它，以节约内存。

4.1.1　载入库

用 loadlibrary 函数将共享库载入到 MATLAB 中。该函数的语法为：

```
loadlibrary('shrlib', 'hfile')
```

其中 shrlib 为 dll 共享库文件的名称，hfile 为包含函数原型的头文件的名称。

作为实例，可以用 loadlibrary 函数载入定义 MATLAB mx 函数的 libmx 库。下面的第 1 条语句指定头文件 matrix.h 的目录，第 2 条语句从 libmx.dll 中载入库，同时指定头文件。

```
hfile = [matlabroot '\extern\include\matrix.h'];
loadlibrary('libmx', hfile)
```

4.1.2　卸载库

用 unloadlibrary 函数卸载库，清空它所占用的内存。例如

```
unloadlibrary libmx
```

4.2　获取库的信息

可以用下面两个函数获取库中函数的信息。

```
libfunctions('libname')
libfunctionsview('libname')
```

两个函数的主要差别在于，libfunctions 函数将信息显示在 MATLAB 命令窗口，并且可以将输出指定给一个变量，而 libfunctionsview 函数在新窗口中用图形显示信息。

使用 libfunctions 函数，将库名作为惟一参数，可以查看 libmx 库中可以获取的函数。注意，不指定输出变量时可以使用 MATLAB 命令语法。

```
libfunctions libmx

Methods for class lib.libmx:
mxAddField                  mxGetFieldNumber   mxIsLogicalScalarTrue
mxArrayToString             mxGetImagData      mxIsNaN
mxCalcSingleSubscript       mxGetInf           mxIsNumeric
mxCalloc                    mxGetIr            mxIsObject
mxClearScalarDoubleFlag     mxGetJc            mxIsOpaque
mxCreateCellArray           mxGetLogicals      mxIsScalarDoubleFlagSet
        .                       .                   .
        .                       .                   .
```

使用 libfunctions 函数的-full 选项，可以列出函数的所有形式，显示的是调用 C 函数的 MATLAB 语法。用于参数列表和返回值的数据类型与 MATLAB 类型而不是 C 类型相匹配。

```
libfunctions libmx -full

Methods for class lib.libmx:
[mxClassID, MATLAB array] mxGetClassID(MATLAB array)
[lib.pointer, MATLAB array] mxGetData(MATLAB array)
[MATLAB array, voidPtr] mxSetData(MATLAB array, voidPtr)
[lib.pointer, MATLAB array] mxGetPr(MATLAB array)
[MATLAB array, doublePtr] mxSetPr(MATLAB array, doublePtr)
uint8 mxIsFinite(double)
uint8 mxIsInf(double)
        .
        .
```

libfunctionsview 函数创建一个新的窗口，显示指定库中定义的所有函数。对于每种方法，显示表 4-1 中的信息。

表 4-1　方法的描述项

表　头	描　述
返回类型	方法返回值的数据类型
名称	函数名称
参数	输入参数的合法数据类型
继承自	与共享库函数无关

下面的命令打开图 4-1 所示的窗口，显示 libmx 库的内容。

```
libfunctionsview libmx
```

图 4-1　显示 libmx 库的内容

跟 libfunctions 函数一样，显示在这里的数据类型是 MATLAB 类型的。

4.3 调用库函数

共享库载入到 MATLAB 中以后，可用 calllib 函数调用库中的任何函数。下面指定库名、函数名和传递给函数的任何参数。

calllib('libname', 'funcname', arg1, ..., argN)

下面的例子从 libmx 库中调用函数来测试保存在 y 中的值。

```
hfile = [matlabroot '\extern\include\matrix.h'];
loadlibrary('libmx', hfile)

y = rand(4, 7, 2);

calllib('libmx', 'mxGetNumberOfElements', y)
ans =
     56

calllib('libmx', 'mxGetClassID', y)
ans =
   mxDOUBLE_CLASS
```

4.4 传递参数

要确定传递参数给库函数时使用什么数据类型，可察看 libfunctionsview 或 libfunctions -full 的输出。这些函数列出特定库中满足参数指定数据类型的所有函数。

MATLAB 提供了一个名称为 shrlibsample 的外部库实例。该库的.dll 文件位于 extern\examples\shrlib 目录下。使用 shrlibsample 库，需要首先用下面的命令将该目录添加到 MATLAB 路径中。

addpath([matlabroot '\extern\examples\shrlib'])

或者用下面的命令使之成为当前工作目录

cd([matlabroot '\extern\examples\shrlib'])

下面的例子载入 shrlibsample 库，并显示库中的一些函数。

```
loadlibrary shrlibsample shrlibsample.h
libfunctions shrlibsample -full

doublePtr multDoubleArray(doublePtr, int32)
double addMixedTypes(int16, int32, double)
[double, doublePtr] addDoubleRef(double, doublePtr, double)
[string, string] stringToUpper(string)
string readEnum(Enum1)
double addStructFields(c_struct)
[lib.pointer, doublePtr] multDoubleRef(doublePtr)
[double, c_structPtr] addStructByRef(c_structPtr)
c_structPtrPtr allocateStruct(c_structPtrPtr)
voidPtr deallocateStruct(voidPtr)
int16Ptr multiplyShort(int16Ptr, int32)
```

这些函数都是用 C 写的。这里看到的是调用 C 函数的 MATLAB 语法。

观察上面列出的函数可以发现，函数的输入输出参数有一些有趣的现象：

● 很多参数类型（如 int32,double）与 C 的对应类型很相似。此时可以直接传递这些 MATLAB 数据类型。

● 有些 C 中的参数类型（如**double, 预定义结构）与标准的 MATLAB 数据类型区别很大。此时通常需要选择，确定是传递标准的 MATLAB 类型还是让 MATLAB 自动转换，或者用 libstruct 和 libpointer 之类的 MATLAB 函数转换数据。

● C 输入参数常常以引用方式传递。尽管 MATLAB 不支持引用传递，但可以创建与 C 引用兼容的 MATLAB 参数。

● C 函数常常将数据放在输入参数中通过引用传递返回。MATLAB 创建额外的输出参数来返回这些值。注意，上面列表中所有以 Ptr 或 PtrPtr 结尾的输入参数也作为输出列出。

传递参数通常有下面几个注意事项：

● 在库函数中以引用方式传递非标量参数时，必须进行声明。

● 如果库函数使用单个子脚本索引来引用一个二维矩阵，记住 C 程序按行处理矩阵，而 MATLAB 按列处理矩阵。所以调用函数以前将输入矩阵进行转置，输出时再进行转置。

● 传递二维以上的数组，数组的形状可能会被 MATLAB 改变。为了确保数组的形状不变，调用函数以前保存数组的大小，然后使用相同大小对输出数组进行重塑来校正数组的维。

```
vs = size(vin)                    % 保存初始维
vs =
     2     5     2

vout = calllib('shrlibsample','multDoubleArray', vin, 20);
size(vout)                        % 维被改变
ans =
     2    10

vout = reshape(vout, vs);         % 把数组保存为 2*5*2
size(vout)
ans =
     2     5     2
```

● 使用空数组[]，将 NULL 参数传递给支持可选输入参数的库函数。

外部库中的很多函数使用引用方式传递的参数。为了使用户可以处理这些函数，MATLAB 传递指针对象给这些参数。

4.5 数据转换

本小节介绍在 MATLAB 中如何转换参数的数据类型，以及用户认为转换数据可以提高效率时如何自己转换数据。

在大多数情况下，MATLAB 会自动对传入和传出外部库函数的数据的类型进行转换，转换为外部函数所希望的类型。但是，有时可能会有选择地手工转换某些参数数据。这样做的好处如下所述。

● 将同一段数据传递给一系列库函数时，开始时就用手工一次进行转换比用 MATLAB

每次调用时都自动转换更有意义。这将省去不必要的复制和转换操作，从而节省时间。

● 传递较大的结构时，可以通过创建与外部函数中使用的 C 结构形状相匹配的 MATLAB 结构来节约内存。libstruct 函数创建与库中 C 结构类似的 MATLAB 结构。

● 外部函数的参数使用一级以上的引用时，比如 double**，将需要传递引用。用 libpointer 函数进行创建。手工创建比 MATLAB 自动创建要好。

4.5.1 简单类型

共享库接口支持所有标准的标量 C 数据类型。简单类型及其等价的 MATLAB 类型，如表 4-2 和表 4-3 所示。MATLAB 使用右列的数据类型，参数的 C 数据类型显示在左列。表 4-3 显示了扩展的 MATLAB 数据类型。它们是 MATLAB 的 lib.pointer 类的实例，不是标准 MATLAB 数据类型。

表 4-2 简单 C 类型及其对应的 MATLAB 类型

C 类型（32 位计算机上）	等价的 MATLAB 类型	C 类型（32 位计算机上）	等价的 MATLAB 类型
char,byte	int8	unsigned int,unsigned long	uint32
unsigned char,byte	uint8	float	single
short	int16	double	double
unsigned short	uint16	char *	string（1*n 的字符数组）
int,long	int32		

表 4-3 C 指针数据类型及扩展的 MATLAB 类型

C 类型（32 位计算机上）	扩展的 MATLAB 类型	C 类型（32 位计算机上）	扩展的 MATLAB 类型
interger 指针类型(int *)	(u)int(size)Ptr	mxArray *	MATLAB 数组
float *	singlePtr	void *	voidPtr
double *	doublePtr	type **	在 typePtr 后面加上 Ptr，如 double **为 doublePtrPtr

1. 转换为其他简单类型

对于简单类型，MATLAB 会自动将参数转换为外部函数所希望的数据类型。这说明，如果一个函数希望接受一个 byte(8 字节整数)型参数，但实际上传递了 double 型参数时 MATLAB 会自动进行转换。例如，下面 C 函数的参数类型为 short,int 和 double。

```
double addMixedTypes(short x, int y, double z)
{
    return (x + y + z);
}
```

可以简单地将所有参数的类型设置为 double 型，MATLAB 会确定每个参数所期望的数据类型并进行合适的转换。

```
calllib('shrlibsample', 'addMixedTypes', 127, 33000, pi)
ans =
  3.3130e+004
```

2. 转换为引用

当外部函数原型定义参数为引用类型时，MATLAB 也会自动将值方式传递的参数转换为引用方式传递的参数。所以将 MATLAB double 型参数传递给希望输入为 double *的函数时，

MATLAB 将它转换为 double 型引用。

addDoubleRef 是 C 函数，要求输入参数的类型为 double *。

```
double addDoubleRef(double x, double *y, double z)
{
    return (x + *y + z);
}
```

用 3 个 double 型参数调用该函数时，MATLAB 进行转换。

```
calllib('shrlibsample', 'addDoubleRef', 1.78, 5.42, 13.3)
ans =
    20.5000
```

3. 字符串

对于参数类型为 char *的情况，可以传递 MATLAB 字符串，即字符数组。下面的 C 函数要求输入参数的类型为 char *。

```
char* stringToUpper(char *input) {
    char *p = input;

    if (p != NULL)
        while (*p!=0)
            *p++ = toupper(*p);
    return input;
}
```

libfunctions 函数显示可以用 MATLAB string 类型作为输入。

```
libfunctions shrlibsample -full
    [string, string] stringToUpper(string)
```

创建一个 MATLAB 字符数组 str，并作为输入参数进行传递。

```
str = 'This was a Mixed Case string';
calllib('shrlibsample', 'stringToUpper', str)
ans =
    THIS WAS A MIXED CASE STRING
```

注意：尽管 MATLAB 传递给 stringToUpper 的输入参数表示一个 char 类型的引用，但它不是真正的引用数据类型。即，它不包含 MATLAB 字符数组 str 的地址。所以，函数执行时，返回正确的结果但不修改 str 中的值。如果现在检查 str，可以发现它的值没有改变。

```
str
str=
  This was a Mixed Case string
```

4.5.2 枚举类型

对于定义为 C 枚举类型的参数，可以传递枚举字符串或它的等价整数。shrlibsample 库中的 readEnum 函数返回与传入参数匹配的枚举字符串。下面是 Enum1 的定义，以及 C 函数 readEnum。

```
enum Enum1 {en1 = 1, en2, en4 = 4} TEnum1;

char* readEnum(TEnum1 val) {
```

```
    switch (val) {
    case 1 :return "You chose en1";
    case 2: return "You chose en2";
    case 4: return "You chose en4";
    default : return "enum not defined";
    }
}
```

在 MATLAB 中，可以将枚举类型表示为枚举字符串或它的等价数值。上面的 Tenum1 定义声明枚举 en4 等于 4。首先用一个字符串调用 readEnum 函数。

```
calllib('shrlibsample', 'readEnum', 'en4')
ans =
    You chose en4
```

现在用等价的数值参数 4 调用它。

```
calllib('shrlibsample', 'readEnum', 4)
ans =
    You chose en4
```

4.5.3 结构

对于参数为结构的库函数，需要传递结构，该结构的字段名与库中结构的相同。要确定名称和结构字段的数据类型，进行下面任意一种操作。

- 参考该库提供的文档。
- 在头文件中查找结构的定义。

也可以从 MATLAB 内部确定外部定义的结构的字段名。创建和初始化结构时，不必匹配数值型字段的数据类型。MATLAB 会在用户用 calllib 函数进行调用时进行正确的转换。按照下面的步骤获取结构的字段名。

- 用 libfunctionsview 函数显示库中所有函数的形式。该函数显示每个函数用到的结构的名称。例如，键入

```
libfunctionsview shrlibsample
```

MATLAB 打开一个显示 shrlibsample 库中函数的形式的新窗口。下面的命令行显示 addStructFields 函数。

```
double addStructFields(c_struct)
```

- 如果当前函数有一个结构参数，从 libfunctionsview 显示中获取结构类型，并调用 libstruct 函数。libstruct 函数返回一个对象，该对象按照库中定义的结构进行构建。

```
s=libstruct('c_struct');
```

- 从 libstruct 返回的对象中获取结构字段的名称。

```
get(s)
    p1:0
    p2:0
    p3:0
```

- 用要传递给库函数的值初始化字段，并用 calllib 进行调用。

```
s.p1=476;s.p2=-299;s.p3=1000;
calllib('shrlibsample', 'addStructFields', s)
```

1. 指定结构的字段名

下面是几个适用于将结构传递给外部库函数的事项：

- 这些字段包含的字段数可能比库结构中定义的少。MATLAB 将任何未定义或未初始化的字段设置为 0。
- 使用的任何结构字段名必须与库结构中的字段名相匹配。注意结构字段名是有大小写区分的。
- 结构不能包含库结构中没有定义的字段。

2. 传递 MATLAB 结构

对于其他数据类型，当外部函数有一个结构参数时（例如 C 结构），可以传递一个 MATLAB 结构给该函数。结构字段名必须与库中定义的字段名相匹配，但是数值型字段的数据类型不用匹配。MATLAB 将 MATLAB 结构的每个数值型字段转换为正确的数据类型。

下面的例子演示 MATLAB 结构的传递。示例共享库 shrlibsample 定义下面的 C 结构和函数。

```
struct c_struct {
    double p1;
    short p2;
    long  p3;
};

double addStructFields(struct c_struct st)
{
    double t = st.p1 + st.p2 + st.p3;
    return t;
}
```

下面的代码传递 MATLAB 结构 sm 给 addStructFields，sm 结构有 3 个 double 型字段。MATLAB 将字段转换为 C 结构 c_struct 中定义的 double,short 和 long 数据类型。

```
sm.p1 = 476;    sm.p2 = -299;    sm.p3 = 1000;
calllib('shrlibsample', 'addStructFields', sm)
ans =
        1177
```

3. 传递 libstruct 对象

传递结构给外部函数时，MATLAB 确保传递的结构与库中定义的相匹配。必须包含为该类型定义的所有必需字段，并且每个字段必须是所要求的数据类型。对于结构中的任何缺失字段，MATLAB 创建一个该名称的空字段，将其值初始化为 0。对于数据类型与结构定义不匹配的字段，MATLAB 将该字段的数据转换为要求的类型。

操作小结构时，让 MATLAB 自动转换就足够了。可以用 calllib 传递初始 MATLAB 结构并让 MATLAB 自动控制转换。但是，操作中多次进行传递一个或多个大结构的调用时，对外部函数作任何调用以前手工转换结构更有利。使用这种方法只需要在开始转换一次结构数据，不需要每次调用函数时都转换，从而可以节省时间。如果转换后的结构字段占用的空间比开始的少，也可以节约内存。

可以用一个步骤将 MATLAB 结构转换为类似的 C 结构。调用 libstruct 函数，传入库中结构的名称和 MATLAB 原始结构就可以了。libstruct 函数的语法格式为

```
s=libstruct('structtype', mlstruct)
```

函数返回的值 s 称为 libstruct 对象。尽管它确实是一个 MATLAB 对象，但它操作起来更像是一个 MATLAB 结构。这个新"结构"的字段是从上面语法中 structtype 指定的外部结构类型获得的。

例如，将 MATLAB 结构 sm 转换为 libstruct 对象 sc，使用下面的语句

```
sm.p1 = 476;    sm.p2 = -299;    sm.p3 = 1000;
sc = libstruct('c_struct', sm);
```

sc 是从 c_struct 结构类型获得的。

初始结构 sm 的字段都是 double 型的。sc 的字段与 c_struct 结构类型的相匹配。这些字段的数据类型分别为 double,short 和 long。

注意，只能将 libstruct 函数用于标量结构。

也可以通过调用只有一个 structtype 参数的 libstruct 函数来创建空的 libstruct 对象。这会创建一个具有所有必需字段的对象，每个字段的值初始化为 0。

```
s=libstruct('structtype')
```

4. 把结构用作对象

libstruct 函数的返回值不是真正的 MATLAB 结构。它实际上是 lib.c_struct 类的实例，通过查看 whos 函数的输出可以看到。

```
whos sc
  Name      Size                    Bytes  Class
  sc        1x1                            lib.c_struct

Grand total is 1 element using 0 bytes
```

本结构的字段作为 lib.c_struct 类的属性来实现。可以用 MATLAB 的面向对象操作函数 set 和 get 读取和修改这些字段中的任何一个，例如：

```
sc = libstruct('c_struct');
set(sc, 'p1', 100, 'p2', 150, 'p3', 200);
get(sc)
    p1: 100
    p2: 150
    p3: 200
```

也可以通过简单地把这些字段视为其他任何 MATLAB 结构字段来读取和修改。

```
sc.p1 = 23;
sc.p1
ans =
    23
```

5. 传递 libstruct 对象示例

仍然使用 addStructFields 实例，这一次在调用函数以前将结构转换为 c_struct 类型的对象。

```
sm.p1 = 476;    sm.p2 = -299;    sm.p3 = 1000;
sc = libstruct('c_struct', sm);
get(sc)
    p1: 476
    p2: -299
    p3: 1000
```

现在调用函数，传递结构 sc。

```
calllib('shrlibsample', 'addStructFields', sc)
ans =
      1177
```

4.5.4 创建引用

可以用传值方式将大部分参数传给外部函数，即使在函数原型声明参数为引用时也是如此。但是，你可能多次发现，将 MATLAB 参数作为 C 引用的等价形式进行传递很有用。

1．使用 libpointer 函数

使用下面的语法，用 libpointer 函数创建引用。

```
p = libpointer('type', 'value')
```

例如，创建一个指向 int16 型值的指针 pv。在 libpointer 函数的第 1 个参数中，输入正在创建的指针类型 type，类型名称总是数据类型（本例中为 int16），后缀为 Ptr。

```
v = int16(485);
pv = libpointer('int16Ptr', v);
```

返回值 pv 实际上是 MATLAB 类 lib.pointer 的实例。lib.pointer 类具有 Value 和 DataType 属性。可以用 MATLAB 的 get 和 set 函数读取和修改这些属性：

```
get(pv)
         Value: 485
      DataType: 'int16Ptr'
```

lib.pointer 类还有两个方法，即 setdatatype 和 reshape。

```
methods(pv)
Methods for class lib.pointer:
   setdatatype    reshape
```

2．创建简单类型的引用

下面是一个演示如何创建和传递 double 型指针，以及如何解释输出数据的简单实例。函数 multDoubleRef 有一个输入，为 double 型引用，返回相同类型的值。

下面是 C 函数：

```
double *multDoubleRef(double *x)
{
     *x *= 5;
     return x;
}
```

创建输入数据 x 的引用 xp，并确认它的值。

```
x = 15;
xp = libpointer('doublePtr', x);
get(xp)
         Value: 15
      DataType: 'doublePtr'
```

现在调用函数并检查结果。

```
calllib('shrlibsample', 'multDoubleRef', xp);
get(xp, 'Value')
```

```
ans =
    75
```

用 libfunctions 函数及其-full 选项查看 shrlibsample 函数的原型，发现 MATLAB 返回 2 个输出。第 1 个为 lib.pointer 类的对象，第 2 个为 doublePtr 输入参数的值属性。

```
libfunctions shrlibsample -full
    [lib.pointer, doublePtr] multDoubleRef(doublePtr)
```

多次运行本实例，然后检查返回的输出值。

```
x = 15;
xp = libpointer('doublePtr', x);

[xobj, xval] = calllib('shrlibsample', 'multDoubleRef', xp)
xobj =
    lib.pointer
xval =
     75
```

与输入的引用参数类似，第 1 个输出 xobj 是一个 lib.pointer 类的对象。可以查看本输出，但首先需要初始化它的数据类型和大小。用 lib.pointer 类定义的 setdatatype 方法将数据类型设置为 doublePtr，大小设置为 1*1。

初始化以后，就可以查看 xobj 的输出。

```
setdatatype(xobj, 'doublePtr', 1, 1)
get(xobj)
ans =
        Value: 75
     DataType: 'doublePtr'
```

第 2 个输出 xval 是从输入 xp 的值复制来的 double 型值。

3. 创建结构引用

创建结构引用参数与创建简单类型的引用参数区别不大。下面函数有一个 c_struct 类型结构的引用参数。返回的输出参数为结构中各字段值的和。该函数还修改输入参数的字段值。

```
double addStructByRef(struct c_struct *st)
{
    double t = st->p1 + st->p2 + st->p3;
    st->p1 = 5.5;
    st->p2 = 1234;
    st->p3 = 12345678;
    return t;
}
```

尽管本函数希望接收一个结构引用类型的输入，但传递结构本身，然后让 MATLAB 将它转换为引用更为容易。下面的例子传递一个 MATLAB 结构 sm 给函数 addStructByRef。MATLAB 将正确值返回到输出 x 中，但不修改输入 sm 的内容，因为 sm 不是引用。

```
sm.p1 = 476;    sm.p2 = -299;    sm.p3 = 1000;
x = calllib('shrlibsample', 'addStructByRef', sm)
x =
        1177
```

也可以用引用方式传递结构。这时函数接收一个指向结构的指针，然后可以修改结构的字段。

```
sp = libpointer('c_struct', sm);
calllib('shrlibsample', 'addStructByRef', sp)
ans =
          1177

get(sp, 'Value')
ans =
     p1: 5.5000
     p2: 1234
     p3: 12345678
```

4.5.5 引用指针

这里所说的引用指针指的是有一个以上引用级别（如 uint16 **）的参数。在 MATLAB 中，这些参数类型的名称都有 PtrPtr 后缀（例如 uint16PtrPtr）。

调用输入参数为引用指针的函数时，可以使用引用参数替换引用指针，然后让 MATLAB 将它转换为引用指针。例如，外部的 allocateStruct 函数希望有一个 c_structPtrPtr 参数。

```
libfunctions shrlibsample -full
   c_structPtrPtr allocateStruct(c_structPtrPtr)
```

下面为 C 函数。

```
void allocateStruct(struct c_struct **val)
{
     *val=(struct c_struct*) malloc(sizeof(struct c_struct));
     (*val)->p1 = 12.4;
     (*val)->p2 = 222;
     (*val)->p3 = 333333;
}
```

尽管原型说明需要 c_structPtrPtr 输入参数，但还是可以使用 c_structPtr 并让 MATLAB 进行转换。下面创建一个空结构的引用参数，并将它传递给 allocateStruct 结构。

```
sp = libpointer('c_structPtr');
calllib('shrlibsample', 'allocateStruct', sp)

get(sp)
ans =
         Value: [1x1 struct]
     DataType: 'c_structPtr'

get(sp, 'Value')
ans =
     p1: 12.4000
     p2: 222
     p3: 333333
```

完成以后，返回分配的内存。

```
calllib('shrlibsample', 'deallocateStruct', sp)
```

第 5 章　DDE（动态数据交换）编程

MATLAB 提供了一些实现动态数据交换(DDE)的函数，利用它们，Windows 应用程序能够通过数据交换实现相互通信，即使得 MATLAB 能够调用其他的 Windows 应用程序或让其他应用程序调用 MATLAB 程序。

5.1　DDE 概念和技巧

应用程序通过 DDE 转换来实现相互通信。转换的发起方应用程序称为客户端，转换的响应方应用程序称为服务器端。当客户端应用程序发起 DDE 转换时，它必须确认两个 DDE 参数，这两个参数由服务器端提供，分别是服务器端应用程序的名称（服务名）和转换的主题（主题）。当服务器端应用程序接收到一个指定主题的转换请求时，它回应请求并建立一个 DDE 转换。

服务器名和主题一起惟一地确认一次转换。在转换过程中，不能对它们进行改变，即使服务器端可以有一个以上的转换也不能。在 DDE 转换过程中，客户端应用程序和服务器端应用程序就彼此关心的项目交换数据。在转换过程中，项目不能改变。下面更详细地介绍相关概念。

服务名：每个可以作为服务器的应用程序都有一个惟一的服务名。服务名通常是应用程序的可执行程序的文件名去掉扩展名以后的名称。下面是一些通用的服务名：MATLAB 的服务名为 MATLAB，Word 的服务名为 WinWord， Excel 的服务名为 Excel。参考应用程序文档，可以得到其他应用程序的服务名。

主题：它定义 DDE 转换的主题，通常对于客户端和服务器端的应用程序都是有意义的。MATLAB 的主题是 Symtem 和 Engine，后面将进行讨论。大部分应用程序支持 System 主题和至少一个其他主题。

项目：每个主题支持一个或多个项目。项目确定 DDE 转换过程中传递的数据。项目名称是否区分大小写与应用程序有关。如果 MATLAB 引擎项目引用矩阵，则它们是区分大小写的，因为矩阵名称是区分大小写的。

剪贴板格式：为了在应用程序之间传递格式化数据，DDE 使用 Windows 剪贴板格式。作为客户端，MATLAB 只支持文本格式。作为服务器端，MATLAB 支持文本、Metafilepict 和 XLTable 等格式。

5.2　MATLAB 作为服务器

客户端应用程序可以通过一些方式来获取作为 DDE 服务器端的 MATLAB，这些方式与客户应用程序有关：如果使用进行 DDE 转换的应用程序，而该应用程序提供了函数或宏，就可以使用这些函数或宏。例如，Excel、Word 和 Visual Basic 就提供了 DDE 函数或宏，可参见相关应用程序的文档得到详细的信息。如果在创建自己的应用程序，可以直接使用 MATLAB

引擎库或 DDE。图 5-1 演示了 MATLAB 作为一个服务器端如何进行通信。客户端应用程序中的 DDE 函数与 MATLAB DDE 服务器端模块进行通信。客户端的 DDE 函数可以通过应用程序或 MATLAB 引擎库提供。

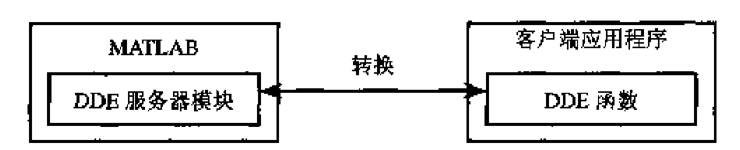


图 5-1　MATLAB 作为服务器端时的通信方式

5.2.1　DDE 命名层次

作为服务器端获取 MATLAB 时，必须指定它的服务名、主题和项目。图 5-2 显示了 MATLAB 的命名层次。

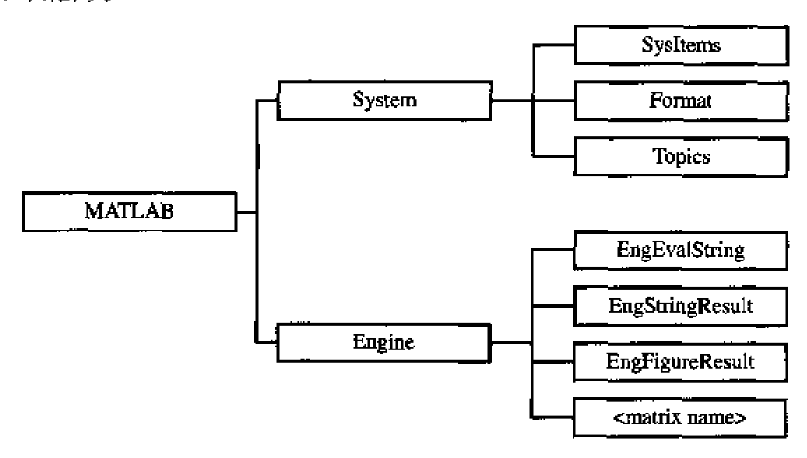


图 5-2　MATLAB 的命名层次

5.2.2　MATLAB 主题和项目

MATLAB 有两个主题，分别为 System 和 Engine。

1．MATLAB 的 System 主题

System 主题允许用户浏览服务器端提供的主题列表、System 主题项目和服务器端所支持的格式。MATLAB 的 System 主题支持下面的项目：

① Sysitems：提供一个 System 主题所支持的项目列表，列表采用表格分隔方式。

② Format：提供所有由服务器端支持的格式的字符串名称列表，采用表格分隔方式。MATLAB 支持 Text、Metafilepict 和 XLTabel 等格式。

③ Topic：提供一个 MATLAB 支持的主题名称的列表。

2．MATLAB 的 Engine 主题

Engine 主题让用户通过提供一个命令来运行 MATLAB、导入和导出数据，这样，MATLAB 作为服务器端使用。

MATLAB 的 Engine 主题支持下面这些选项：

- EngEvalString：在传递给 MATLAB 一个命令时，如果需要的话，指定一个项目名称。
- EngStringResult：从 MATLAB 导入数据时提供 DDE execute 命令的字符串结果。
- EngFigureResult：从 MATLAB 导入数据时提供 DDE execute 命令的图形结果。

- <matrix name>：从 MATLAB 导入数据时，数据矩阵的名称。

MATLAB 的 Engine 主题支持三种操作，它们可以通过 DDE 客户端界面被应用程序使用。这些操作包括给 MATLAB 传递命令、从 MATLAB 导入数据和传递数据给 MATLAB。

（1）给 MATLAB 传递命令

客户端应用程序可以用 DDE 的 execute 命令向 MATLAB 传递命令。Engine 主题用两种方式支持 DDE 的运行，因为有的客户端程序需要指定项目名称和要运行的命令，而有的客户端程序则只需要提供命令。需要用到项目名称时，使用 EngEvalString。两种形式都要求命令的格式为文本格式。DDE execute 参数的描述如表 5-1 所示。

表 5-1　DDE execute 参数描述

项　目	格　式	命　令
EngEvalString	Text	String
Null	Text	String

（2）从 MATLAB 导入数据

使用 DDE 的 request 命令从 MATLAB 导入数据。Engine 主题对三个函数支持 DDE 请求。

① 前一个 DDE execute 命令的文本结果：使用文本格式的 EngStringResult 项目导入 DDE execute 命令的字符串结果。

② 前一个 DDE execute 命令的图形结果：使用 EngFigureResult 项目导入 DDE execute 命令的图形结果。EngFigureResult 项目可以有 Text 和 Metafilepict 两种格式：

用值为“yes”或“no”的字符串指定 Text 格式的结果。如果结果为“yes”，当前图形的图元文件被放置到剪贴板上。这种格式提供给只从 DDE 请求中获取文本的 DDE 客户程序，如 Word for Windows 等。如果结果为“no”，不在剪贴板上放置任何图元文件。

当有图形结果导致一个图元文件直接从 DDE 请求中返回时，指定为 Metafilepict 格式。

③ 指定矩阵的数据：通过将矩阵名称指定为项目来导入矩阵数据。可以指定为 Text 格式或 XLTable 格式。表 5-2 总结了 DDE 请求的参数。

表 5-2　DDE 请求的参数

项　目	格　式	结　果
EngStringResult	Text	字符串
EngFigureResult	Text	Yes/No
EngFigureResult	Metafilepict	当前图片的图元文件
<matrix name>	Text	字符缓冲器，列表间隔
<matrix name>	XLTable	二进制数据，与 Excel 兼容

（3）传递数据给 MATLAB

客户程序使用 DDE 的 poke 命令将数据传递给 MATLAB。Engine 主题支持 DDE 的 poke 命令在 MATLAB 工作空间中更新和创建新矩阵。指定的项目为要更新或创建的矩阵的名称。如果指定名称的矩阵在工作空间中已经存在，它将被更新；如果不存在，将创建一个新的矩阵。矩阵数据可以有 Text 或 XLTable 两种格式。下表概述了 DDE poke 命令的参数：

表 5-3　DDE poke 命令参数

项　目	格　式	结　果
<matrix name>	Text	字符缓冲器，列表间隔
<matrix name>	XLTabl	二进制数据，与 Excel 兼容

5.3 MATLAB 作为客户

要将 MATLAB 作为客户端应用程序，使用 MATLAB 的 DDE 客户函数建立和保持转换。

5.3.1 相关函数

MATLAB 作为 DDE 客户端时包含一系列函数，它们允许用户将 MATLAB 作为客户端程序使用。这些函数如表 5-4 所示。

表 5-4　DDE 函数

函　数	描　述
ddeadv	安装 MATLAB 与 DDE 服务器端应用程序之间的提示链接
ddeexec	给 DDE 服务器端应用程序传送运行字符串
ddeinit	初始化 MATLAB 与另一个应用程序之间的 DDE 转换
ddepoke	从 MATLAB 向 DDE 服务器端应用程序传送数据
ddereq	请求源于 DDE 服务器端应用程序的数据
ddeterm	终止 MATLAB 与服务器端应用之间的 DDE 转换
ddeunadv	取消 MATLAB 与 DDE 服务器端应用程序之间的提示链接

下面给出这些函数的调用格式及说明。

1．ddeadv 函数

该函数在 MATLAB 与服务器应用程序之间创建 advisory 链接，调用格式为：

```
rc = ddeadv(channel,'item','callback')
rc = ddeadv(channel,'item','callback','upmtx')
rc = ddeadv(channel,'item','callback','upmtx',format)
rc = ddeadv(channel,'item','callback','upmtx',format,timeout)
```

item 参数识别的数据发生变化时，callback 参数指定的字符串传递给 eval 函数并进行处理。如果 advisory 链接不是热链接，则 DDE 修改更新矩阵 upmtx 来反映 item 中的数据情况。

如果忽略不在参数列表末尾的可选参数，则必须用空矩阵表示缺失参数。如果成功，ddeadv 函数在变量 rc 中返回 1，否则返回 0。各参数的说明如表 5-5 所示。

表 5-5　参数说明

参　数	描　述
channel	ddeinit 函数建立的会话通道
item	为 advisory 链接指定 DDE 选项名称的字符串。改变 item 表示的数据会触发 advisory 链接
callback	为指定更新通知时进行处理的回调的字符串。改变 item 确定的数据会导致 callback 参数传给 eval 函数进行处理
upmtx（可选）	为字符串，该字符串指定其数据与更新通知一起发送的矩阵的名称。如果包含 upmtx 参数，改变 item 参数会导致 upmtx 参数被修订数据更新。指定 upmtx 参数会创建一个热链接。如果 upmtx 函数存在于工作空间中，则覆盖其内容。如果 upmtx 参数不存在，则创建它

续表

参　数	描　述
format （可选）	为12元素的数组，指定要发送的数据的格式。第1个元素指定使用的Windows剪贴板格式，目前只支持cf_text格式，它对应于值1。第2个元素指定生成的矩阵的类型。合法的类型为numeric型和string型，它们分别对应于值0和值1。默认的格式数组为[1 0]
timeout （可选）	指定该操作终止联接的时间限制，timeout参数用毫秒指定(1000毫秒=1秒)。timeout的默认值为3秒。如果在timeout毫秒内没有建立advisory链接，该函数失败

下面的例子建立Excel单元范围与矩阵x之间的热链接。如果链接成功，则显示矩阵：

```
rc = ddeadv(channel, 'r1c1:r5c5', 'disp(x)', 'x');
```

先前必须已经用ddeinit命令建立起了与Excel之间的通信。

2. ddeexec函数

该函数发送要执行的字符串，调用格式为：

```
rc = ddeexec(channel,'command')
rc = ddeexec(channel,'command','item')
rc = ddeexec(channel,'command','item',timeout)
```

ddeexec函数通过已经建立的DDE会话发送要执行的字符串给另一个应用程序。将该字符串作为command参数进行指定。如果忽略不在参数列表末尾的可选参数，必须用空矩阵替换缺失参数。如果成功，ddeexec在变量rc中返回1，否则返回0。函数参数如表5-6所示。

表5-6　参数列表

参　数	描　述
channel	ddeinit函数建立的会话通道
command	指定要执行的命令的字符串
item （可选）	指定DDE选项名称。很多应用程序不需要该参数。如果需要该参数，它会提供其他信息给command参数
timeout （可选）	指定该操作终止联接的时间限制，timeout参数用毫秒指定(1000毫秒=1秒)。timeout的默认值为3秒

下面的例子假设已经建立了会话通道，给Excel发送一个命令。

```
rc = ddeexec(channel,'[formula.goto("r1c1")]')
```

3. ddeinit函数

该函数初始化DDE会话，调用格式为：

```
channel=ddeinit('service','topic')
```

返回分配给会话的通道，它用在其他MATLAB DDE函数中。'service'是一个字符串，为会话指定服务或应用名。'topic'为指定会话主题的字符串。

下面的例子为电子表格文件'stocks.xls'建立与Excel之间的会话。

```
channel = ddeinit('excel','stocks.xls')
channel =
          0.00
```

4. ddepoke函数

该函数发送数据给应用程序，调用格式为：

```
rc = ddepoke(channel,'item',data)
rc = ddepoke(channel,'item',data,format)
rc = ddepoke(channel,'item',data,format,timeout)
```

ddepoke 函数通过已经建立的 DDE 会话发送数据给应用程序。ddepoke 函数在将数据矩阵发送给服务器程序以前对它进行格式化：

字符串矩阵逐元素进行转换为字符，发送生成的符号缓冲区。

数值矩阵作为列用表格键进行间隔的数字进行发送。只发送非稀疏矩阵的实部。

如果忽略不位于参数列表末尾的可选参数，必须用空矩阵替换缺失参数。如果成功，ddepoke 函数在变量 rc 中返回 1，否则范围 0。

参数 data 表示包含要发送数据的矩阵。

下面的例子假设已经用 ddeinit 函数建立了与 Excel 之间的会话通道。发送一个 5×5 的单位矩阵给 Excel，将第 1 行第 1 列到第 5 行第 5 列的数据放到该矩阵中。

```
rc = ddepoke(channel, 'r1c1:r5c5', eye(5));
```

5．ddereq 函数

该函数从应用程序请求数据，调用格式为：

```
data = ddereq(channel,'item')
data = ddereq(channel,'item',format)
data = ddereq(channel,'item',format,timeout)
```

ddereq 函数通过已经建立的 DDE 会话从服务器应用程序请求数据。该函数返回一个包含请求到的数据的矩阵，如果请求不成功，返回一个空矩阵。

下面的例子假设有一个 Excel 电子表格文件 stocks.xls。该电子表格第 3 行第 1~3 列包含 3 种股票的价格，第 6～8 行第 2 列包含这些股票的持股数。用下面的命令建立与 Excel 之间的会话。

```
channel = ddeinit('excel','stocks.xls')
```

DDE 函数要求用 rxcy 个形式表示 Excel 的电子表格。用 Excel 术语表示时，价格位于 r3c1:r3c3 中，持股数位于 r6c2:r8c2 中。

从 Excel 请求价格数据，使用下面的命令行：

```
prices = ddereq(channel,'r3c1:r3c3')
prices =
    42.50      15.00      78.88
```

用下面的命令行请求每种股票的持股数：

```
shares = ddereq(channel, 'r6c2:r8c2')
shares =
      100.00
      500.00
      300.00
```

6．ddeterm 函数

使用该函数终止 DDE 会话，调用格式为：

```
rc=ddeterm(channel)
```

输入参数为 ddeinit 函数创建的 DDE 会话通道变量。该函数终止该会话。返回值 rc 等于

0 时表示终止失败，等于 1 时表示成功。

下面的例子终止先前用 ddeinit 建立的会话通道。

```
rc = ddeterm(channel)
rc =
      1.00
```

7．ddeunadv 函数

该函数释放 advisory 链接，调用格式为：

rc = ddeunadv(channel,'*item*')

rc = ddeunadv(channel,'*item*',format)

rc = ddeunadv(channel,'*item*',format,timeout)

ddeunadv 函数释放 MATLAB 与前面通过 ddeadv 函数建立的服务器程序之间的热链接。channel,item 和 format 必须与调用 ddeadv 函数时指定的相同。如果包含 timeout 参数，但接受默认的 format 参数，必须将 format 参数指定为空矩阵。

如果成功，ddeunadv 函数在变量 rc 中返回 1，否则返回 0。

下面的例子释放前面用 addadv 建立的 advisory 链接。

```
rc = ddeunadv(channel, 'r1c1:r5c5')
rc =
      1.00
```

5.3.2 DDE 提示链接

当服务器端的数据已经改变时，使用 DDE 通知客户端应用程序。例如，使用 MATLAB 分析 Excel 电子表格中的数据，同样可以通过建立链接来用新的或经过更改的电子表格数据自动更新矩阵。MATLAB 支持两种提示链接，即热链接和暖链接，它们通过服务器端应用程序提示的方式来进行区分。

- 热链接使数据根据项目的改变来定义时，让服务器端给 MATLAB 提供数据。
- 暖链接在数据改变但只在 MATLAB 请求时提供数据的情况下让服务器端给 MATLAB 提供数据。

用 ddeady 和 ddeunadv 两个函数安装和卸载提示链接。MATLAB 只在它是客户端的时候支持链接。

第6章　COM编程

组件对象模型（Component Object Model，COM）是一系列面向对象技术和工具的集合。使用该集合，软件开发人员可以用不同厂商提供的组件集成他们自己的应用程序。MATLAB提供了COM支持，使得可以交互处理控件或服务器进程，或者将MATLAB作为计算服务器，用客户应用程序进行控制。

6.1　MATLAB COM集成简介

COM提供了将可重用的二进制组件集成为应用程序的框架。组件是用编译后的代码实现的，所以其源代码可以用任何支持COM的程序语言编写。组件使得应用程序的更新简化了，因为使用组件不需要重新编译整个程序。另外，对于应用程序来说，组件的位置是透明的，所以可以在不修改应用程序的情况下将组件重新部署到单独的进程甚至远程系统。

使用COM，开发者和终端用户可以选择不同厂商提供的组件并将它们集成到一个完整的应用程序中。例如，一个单独的应用程序可能需要实现数据库访问、数学分析和商业质量图形表示等功能。使用COM，开发人员可以选择某厂商提供的数据库访问组件，另一厂商提供的商业图形组件，然后将它们集成到一个数学分析软件包中。当然，这个数学分析模块也可以用某个厂商提供的组件来实现。

6.1.1　概念和术语

在学习COM编程知识以前，了解和理解相关概念和术语是必要的，也是很重要的。

1. COM对象

COM对象是组件对象类或组件的实例。COM强制完成对象的封装，以防止组件的数据和实现被直接访问。对象的方法必须通过接口来访问。COM对象运行在服务器应用程序上，该服务器应用程序被一个或多个客户端应用程序所控制。

2. 程序标识符

从MATLAB内部创建COM组件的实例时，需要用组件的程序标识符，或ProgID引用它。ProgID是组件提供商定义的字符串，用于惟一定义特定COM组件。可以从厂商提供的文档中获取该标识符。MATLAB的ProgID为matlab.application。

3. 接口

可以通过接口访问COM对象的属性和方法。实际上，访问COM对象内部的惟一途径也就是它的接口。接口不包含任何方法的实现，它只是作为指针指向对象内部的方法。

接口通常将逻辑上相关的方法组合在一起。一个对象可以，而且常常提供多个接口。为了使用COM对象，必须知道它支持什么接口，以及该对象实现的方法、属性和事件。组件常会提供这些信息。

4. IUnknown、IDispatch、定制和双重接口

COM 对象的基本接口类型包括：

- IUnknown 接口：基本的工业标准接口，要求所有 COM 对象都有该接口。所有其他接口都是从该接口衍生而来的。
- IDispatch 接口：工业标准接口，它采用较小的函数集来获取 COM 对象的属性和方法的信息并访问它们。
- 定制接口：自定义的接口，允许客户更直接、更快地访问服务器对象的属性和方法。
- 双重接口：双重接口是 IDispatch 接口和定制接口的组合。

5. ActiveX 控件

ActiveX 控件是一种组件，该组件的用户接口使它可以对用户的行为作出响应。例如，一个有"OK"和"Cancel"按钮的窗口可以用控件实现。控件在它的客户应用程序的进程地址空间中运行。该客户应用程序曾被称为控件容器，因为它包含控件。

客户应用程序可以直接访问控件的方法和属性。控件也必须能够向客户端回传信息。这使得它可以通知客户应用程序，有事件发生了，例如发生了鼠标单击事件。

6. 进程内和进程外服务器

可以将服务器设定为下面三种方式中的一种。MATLAB 支持所有这三种方式。

- 进程内服务器：用动态链接库（DLL）或 ActiveX 控件实现的组件，与客户程序运行在相同的进程中，共享同一地址空间。这一特点使得客户端和服务器端的通信相对简单，并且快速。
- 本地进程外服务器：用可执行程序（.exe）实现的组件。该可执行程序运行在单独的进程内，与客户程序的进程不同，但客户程序和服务器程序的进程位于相同的计算机系统中。这种设置方式比进程内的设置方式慢一些，因为跨进程传递数据需要一些消耗。
- 远程进程外服务器：这是进程外服务器的另一种类型。在这种类型中，客户和服务器进程处于不同的系统上，通过网络进行通信，加上跨进程传递数据的消耗，这种设置方式的速度最慢，只能在支持分布式组件对象模型（DCOM）的系统上使用这种设置方式。

6.1.2 支持的客户/服务器设置

可以将 MATLAB 设置为控件，或用其他 COM 组件进行控制。MATLAB 控制其他组件时，MATLAB 是客户，其他组件是服务器；MATLAB 被其他组件控制时，它充当服务器。

MATLAB 支持 4 种不同的 COM 客户/服务器设置方式，包括：

- MATLAB 客户和进程内服务器；
- MATLAB 客户和进程外服务器；
- 客户程序和 MATLAB 自动化服务器；
- 客户程序和 MATLAB 引擎服务器。

1. MATLAB 客户和进程内服务器

如图 6-1 所示，MATLAB 客户应用程序与用 ActiveX 控件或动态链接库（DLL）实现的组件进行交互。服务器与客户运行在同一进程中，并共享同一地址空间。在客户和服务器之间通信很快，因为在同一进程内传递数据消耗很小。

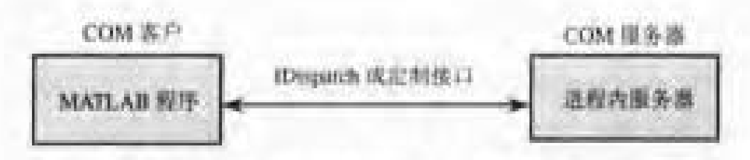


图 6-1　MATLAB 客户与进程内服务器通信

根据组件实现的接口类型，服务器通过 IDispatch 接口或定制接口提供其属性和方法。

ActiveX 控件通常是有某种图形用户界面的对象。MATLAB 在服务器中创建控件时，将控件的 GUI 放在 MATLAB 图形窗口中，使用户可以与它进行交互。通过单击用户界面上的不同选项，可以触发相应的事件。客户确定发生不同事件时如何响应并作出相应的响应。

MATLAB 自己带了一个简单的 ActiveX 控件例子，它在屏幕上画圆，显示一些文本。可以用该控件做一些小试验。

任何已经用动态链接库实现的 COM 组件也初始化为进程内服务器。即与 MATLAB 客户程序在相同的进程中创建。与控件不同，DLL 服务器运行在单独的窗口中，而不是 MATLAB 图形窗口中。

MATLAB 对 DLL 服务器生成的事件的响应方式与 ActiveX 控件的相同。

2. MATLAB 客户和进程外服务器

这种设置方式下，MATLAB 客户应用程序与实现为可执行文件的组件进行交互。该可执行组件在服务器中进行初始化，服务器与客户程序运行在各自的进程中。因为跨进程传递数据，这种设置方式下的速度慢一些。常用的 Excel 和 Word 等可以看作本地服务器。

通信机制如图 6-2 所示。因为客户程序和服务器运行在单独的进程中，可以在同一网络中的任何系统上创建服务器。远程服务器只在支持 DCOM 的环境中设置。

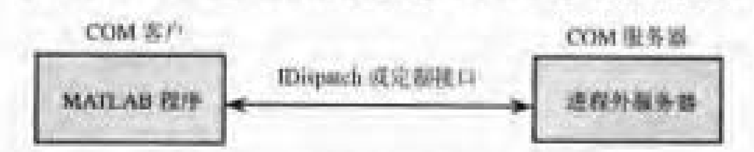


图 6-2　MATLAB 客户和进程外服务器的通信

如果组件提供用户界面，界面显示在单独的窗口中。

3. 客户程序和 MATLAB 自动化服务器

在这种方式中，MATLAB 作为自动化服务器进行操作。任何可以作为自动化控制器的 Windows 程序都可以创建和控制它，通信机制如图 6-3 所示。

可作为自动化控制器的应用程序包括 Microsoft Excel、Microsoft Access、Microsoft Project，以及很多 Visual Basic 和 Visual C++程序。

MATLAB 自动化服务器的功能包括在 MATLAB 工作空间中执行命令，并直接从工作空间获取矩阵和将矩阵放到工作空间。可以用共享和独占模式启动 MATLAB 服务器，还可以选择在本地或远程系统上运行它。



图 6-3　客户程序和 MATLAB 自动化服务器的通信

从外部应用程序创建 MATLAB 服务器，使用该语言的合适函数来初始化服务器。例如，可以使用 Visual Basic 中的 CreateObject 函数。程序标识符指定为 matlab.application。用独占模式运行 MATLAB 服务器，指定标识符为 matlab.application.single。

创建 MATLAB 服务器的函数返回一个句柄。利用这个句柄，通过 IDispatch 接口可以访问服务器中的属性和方法。

4．客户程序和 MATLAB 引擎服务器

MATLAB 为 C，C++或 Fortran 编写的客户应用程序提供了一个速度更快的定制接口 IEngine。MATLAB 使用 IEngine 接口实现客户程序和 MATLAB 引擎服务器之间的通信，通信机制如图 6-4 所示。



图 6-4　客户程序和 MATLAB 引擎服务器之间的通信

MATLAB 提供了 C 引擎函数库和 Fortran 引擎函数库，利用库中的函数可以启动和终止服务器进程，以及传递 MATLAB 处理的命令。

6.1.3　注册控件和服务器

大部分控件和服务器是自动注册的。但是，如果有新的.ocx, .dll 或其他控件或服务器对象文件，必须手工注册这些文件。

使用 DOS 的 regsvr32 命令在 Windows 注册表中注册对象文件（即.ocx 文件）。在 DOS 命令行中，用 cd 函数进入 ocx 文件所在的目录，并使用下面的语法

```
regsvr32 filename.ocx
```

例如，注册很多 MATLAB 文档中都使用了的 mwsamp2.ocx 文件，键入

```
regsvr32 mwsamp2.ocx
```

下面几种方法可以确认控件或服务器是否已经注册（用 MATLAB 的 mwsamp 控件作为例子）。

- 在 Microsoft Visual C++(MSVC) 6.0 编程环境中找到工具菜单，执行 ActiveX 控件测试容器。单击“Edit”按钮，插入新控件，然后选择“MwSamp Control”选项。如果插入控件没有发生问题，说明控件已经注册成功了。注意，这种方法只用于控件。
- 在 DOS 命令行中键入 regedit，打开注册表编辑器。然后可以从“Edit”菜单中选择“Find”选项，搜索控件或服务器对象。它具有类似下面的结构：

```
HKEY_CLASSES_ROOT/progid
```

- 从 MSVC 6.0 的工具菜单中打开 OLE 察看器，你的控件对象应该具有类似下面的结构。

```
Object Classes : Grouped by Component Category : Control :
Your_Control_Object_Name (i.e. Object Classes : Grouped by
Component Category : Control : Mwsamp Control)
```

6.2 MATLAB COM 客户支持

本节介绍用于创建、操作和销毁 COM 控件和服务器对象的 MATLAB 函数。这些对象是 MATLAB COM 类的实例。

6.2.1 创建服务器进程

MATLAB 提供了 2 个函数，让 COM 客户在服务器进程中创建 COM 组件的实例。

1．actxcontrol 函数

该函数在当前图形窗口中创建 ActiveX 控件，调用格式为：

h=actxcontrol('progid')

h=actxcontrol('progid', position)

h=actxcontrol('progid', position, fig_handle)

h=actxcontrol('progid', position, fig_handle, event_handler)

h=actxcontrol('progid', position, fig_handle, event_handler, 'filename')

- h=actxcontrol('progid')：在图形窗口中创建一个 ActiveX 控件。所创建控件的类型由字符串 progid 确定，progid 是控件的程序标识符（ProgID）。返回的对象 h 表示控件的默认接口。
- h=actxcontrol('progid', position)：创建 ActiveX 控件，其位置和大小由矢量 position 指定。该矢量的格式为：

[x y width height]

矢量的前两个元素确定控件在图形窗口中的位置，x 和 y 为控件左下角相对于图形窗口左下角的偏移距离，单位为像素。后面两个元素确定控件的宽度和高度。position 矢量的默认值为[20 20 60 60]。

- h=actxcontrol('progid', position, fig_handle)：在已经存在的图形窗口中，在指定位置 position 处创建一个 ActiveX 控件。该图形窗口用句柄图形对象 fig_handle 表示。默认的图形句柄为 gcf。
- h=actxcontrol('progid', position, fig_handle, event_handler)：创建一个响应事件的 ActiveX 控件。不管什么时候触发事件时，控件通过调用一个 M 文件函数来响应事件。
- h=actxcontrol('progid', position, fig_handle, event_handler, 'filename')：用前 4 个参数创建 ActiveX 控件，将它的初始状态设置为与前面保存的控件的一样。MATLAB 从 filename 指定的文件中载入初始状态。如果不想指定事件控制器参数 event_handler，可以用空矩阵作为第 4 个参数。progid 参数必须与所保存的控件的 progid 相匹配。

event_handler 参数有多种合法格式，用该参数指定下面类型中的一种：

➢ 为控件支持的每个事件指定一个不同的事件控制过程；

➢ 指定一个公共过程来控制选定的事件；

➢ 指定一个公共过程来控制所有事件。

对于第一种情况，用一个单元数组作为 event_handler 参数的值，数组的每一行指定一个事件/控制函数对。

{'event' 'eventhandler'; 'event2' 'eventhandler2'; ...}

event 可以是包含事件名称的字符串，或者作为事件标识符的数值值，eventhandler 是控

件用于控制事件时识别 M 文件函数的字符串，只包括那些要起作用的事件。

对于第 2 种情况，使用与刚才相同的单元数组语法，但是给每个事件指定相同的事件控制过程，还只包括那些要起作用的事件。

对于第 3 种情况，使 event_handler 为包含控制控件所有事件的 M 文件函数的名称的字符串。

在 event_handler 单元数组中没有限定事件/控制过程对的个数。事件控制函数应该接收可变的参数个数。event_handler 参数中使用的字符串没有大小写区分。

2. actxserver 函数

该函数创建 COM 自动化服务器，调用格式为：

```
h=actxserver('progid')
h=actxserver('progid', 'systemname')
```

- h=actxserver('progid')：创建一个 COM 服务器，并返回表示服务器默认接口的 COM 对象 h。prigid 为组件在服务器中进行实例化的程序标识符。该字符串由控件或服务器程序的供应商提供，并且可以从供应商的文档中获得。例如，MATLAB 的 progid 值为 matlab.application。
- h=actxserver('progid', 'systemname')：创建一个运行在远程系统上的 COM 服务器，远程系统的名称由 systemname 参数指定。它可以是一个 IP 地址或 DNS 名称。只在支持分布式组件对象模型（DCOM）的环境中可以使用该语法。

注意：对于实现为动态链接库的组件，actxserver 函数创建一个进程内的服务器。对于实现为可执行程序的组件，actxserver 创建一个进程外的服务器。进程外服务器可以在客户系统或支持 DCOM 的网络上的任何其他系统上创建。

如果控件实现了任何自定义接口，使用 interfaces 函数列出它们，并用 invoke 函数获取选定接口的句柄。目前不支持从自动化服务器创建的事件。

6.2.2 创建 ActiveX 控件

可以用图形用户界面或直接从命令行用 actxcontrol 函数从 MATLAB 客户端创建 ActiveX 控件。这两种方法都在 MATLAB 客户进程中创建控件的一个实例，并给 COM 对象的主接口返回一个句柄。然后，通过该接口获取对象的所有属性或方法。还可以创建额外的接口给对象，包括使用 IDispatch 和任何定制接口的接口。

本小节介绍如何在客户进程中创建控件，以及如何在 MATLAB 图形窗口中放置它的图形界面。

1. 查找已经安装的控件

使用 actxcontrollist 函数可以查看系统上当前安装的 COM 控件。键入

```
list=actxcontrollist;
```

MATLAB 返回每个控件的列表，包括输出的单元数组中的名称、程序标识符（或 ProgID）及文件名。

下面是一些控件可能返回的信息的示例：

```
list = actxcontrollist;
for k = 114:115
    s=sprintf(' Name =  %s\n ProgID =  %s\n File =  %s\n', list{k,:})
end
```

```
s =
    Name = Mwsamp2 Control
    ProgID = MWSAMP.MwsampCtrl.2
    File = D:\R14\bin\win32\mwsamp2.ocx

ans =
    Name = NetMeeting Application
    ProgID = NetMeeting.App.1
    File = C:\WINNT\System32\msconf.dll
```

2．用图形界面创建控件对象

创建控件对象最简单的方法是使用 actxcontrolselect 函数。该函数显示一个图形用户界面，其中列出了安装在系统上的所有控件，如图 6-5 所示。从列表中选择一个选项并单击"Create"按钮，MATLAB 在图形窗口中创建该控件。

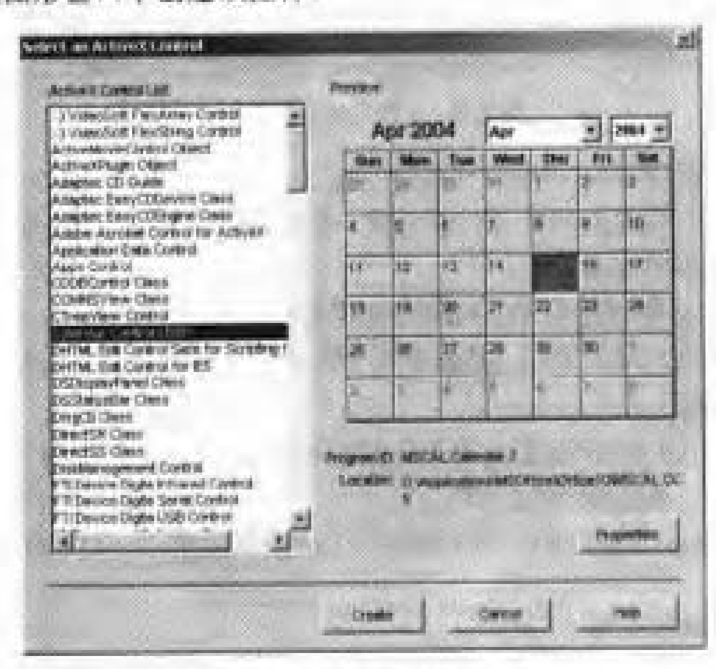


图 6-5　用图形界面创建控件

界面中，左侧有一个选择面板，右侧有一个预览面板。单击选择面板中的控件名，会在预览面板中显示该控件的外观。如果该控件没有预览，则预览面板为空。如果 MATLAB 不能创建控件，则预览面板中显示一个出错消息。

单击窗口中的"Properties"按钮，然后可以在打开的属性窗口中设置控件的各种属性。可以选择控件放在哪个图形窗口中（Parent 文本框），放在窗口中的什么位置（X 和 Y 文本框），以及控件的大小（Width 和 Height 文本框）。

还可以注册任何事件，并用它们控制各种响应。通过在"Callback M-File"下面事件的右侧输入过程名，可以注册事件和回调过程来控制该事件。

还可以通过浏览各自的 M 文件来选择回调过程。单击"Event Names"列中的名称，然后单击"Browse"按钮来选择该事件的回调。将一个回调过程指定给多个事件，按住 Ctrl 键并单击事件名，或者在事件名上连续拖拉鼠标，然后单击"Browse"按钮来选择回调过程。

MATLAB 只对那些注了册的事件作出响应，所以任何没有在"Callback M-File"中指定

回调 M 文件的事件将被忽略。

在 Select an ActiveX Control 窗口中选择“Calendar Control 10.0”，单击“Properties”按钮查看图 6-6 显示的窗口。在“Width”文本框中输入 500，在“Height”文本框中输入 350，改变控件的默认大小。单击“OK”按钮，然后在 actxcontrolselect 窗口中单击“Create”按钮，创建日历控件。

图 6-6　设置日历控件的参数

还可以用 actxcontrol 函数设置控件参数。有一个参数可以用 actxcontrol 设置，但不能用 actxcontrolselect 设置，它是初始化文件的名称。指定该文件名时，MATLAB 将控件的初始状态设置为先前保存的控件的状态。

actxcontrolselect 函数创建 MATLAB COM 类的实例对象。该函数最多返回 2 个输出参数：对象的句柄和一个 1*3 的单元数组，数组中包含控件的有关信息。它们分别用 h 和 info 表示。

```
[h, info] = actxcontrolselect;
```

调用其他 MATLAB COM 函数时使用句柄识别这个特定控件对象。单元数组中返回的信息显示了控件的名称，ProgID 和文件名。

如果选择 Calendar Control 9.0，然后单击“Create”按钮，MATLAB 返回下面的信息：

```
h =
    COM.mscal.calendar.7
info =
     [1x20 char]     'MSCAL.Calendar.7'     [1x41 char]
```

扩展 info 单元数组，显示控件的名称、ProgID 和文件名。

```
info{:}
ans =
    Calendar Control 9.0
ans =
    MSCAL.Calendar.7
ans =
    D:\Applications\MSOffice\Office\MSCAL.OCX
```

3．从命令行创建控件对象

如果知道使用哪个控件并且知道它的 ProgID 值，可以用 actxcontrol 函数创建它。调用该函数需要的输入参数只有 ProgID。但是，像使用 actxcontrolselect 一样，也可以提供其他输入，使得可以选择控件显示在哪个图形窗口中，放在窗口中的哪个位置，以及控件的大小是多少

等。也可以注册事件，或设置控件的初始状态。

actxcontrol 函数返回对象的句柄。进行其他函数调用时用该句柄引用对象。还可以用该句柄获取对象的其他接口。

下面的例子创建一个控件来运行 Microsoft Calendar 应用程序，将该控件放在图形窗口 fig3 中，原点位置为[0,0]，宽度为 300，高度为 400，单位为像素。

```
fig3 = figure('position', [50 50 600 500]);
h = actxcontrol('mscal.calendar', [0 0 300 400], fig3)
h =
    COM.mscal.calendar
```

4．重新设置控件在图形窗口中的位置

创建控件以后，可以用 move 函数改变它在窗口中的大小和位置。对于上面的例子，指定新原点的位置为(70,120)，新的宽度和高度为 400 和 350。

```
h.move([70 120 400 350]);
```

6.2.3 创建 DLL 组件的实例

用 actxserver 函数创建动态链接库实现的服务器组件。MATLAB 在包含客户应用程序的进程中创建组件的实例。

与 DLL 组件一起使用时，actxserver 函数的语法格式为：

```
actxserver(ProgID)
```

其中，ProgID 为组件的程序标识符。

actxserver 函数返回对象句柄。使用该句柄，可以在进行其他 COM 函数调用时引用对象。也可以使用句柄获取对象的其他接口。

与 ActiveX 控件不同，服务器显示的任何用户界面出现在单独的窗口中。

6.2.4 创建 EXE 组件的实例

使用 actxserver 函数，还可以创建用可执行程序实现的服务器组件。但是，这时 MATLAB 是在进程外服务器中创建组件的实例。

actxserver 函数的语法格式为：

```
actxserver(ProgID,sysname)
```

其中，ProgID 为组件的程序标识符，sysname 为可选参数，用于注册已经发布的 COM(DCOM) 系统。

actxserver 函数返回对象句柄。使用该句柄，可以在进行其他 COM 函数调用时引用对象。也可以使用句柄获取对象的其他接口。

下面的例子创建一个运行 Excel 的 COM 服务器应用程序，返回的句柄赋给 h。

```
h = actxserver('excel.application')
h =
    COM.excel.application
```

6.2.5 访问对象的接口

COM 组件可以提供不同类型的接口来访问对象的公共属性和方法。

1. IUnknown 和 IDispatch

调用 actxserver 或 actxcontrol 函数时，MATLAB 创建服务器并返回句柄。MATLAB 按照下面的规则确定返回哪个句柄。

- 首先获取组件 IUnknown 接口的句柄。所有 COM 组件必须至少实现这个接口。
- 然后 MATLAB 试图从组件那里获得 IDispatch 接口。如果组件中实现了 IDispatch 接口，MATLAB 返回该接口的句柄。如果没有实现该接口，MATLAB 返回 IUnknown 接口的句柄。

组件常常基于 IDispatch 接口提供其他接口，它们作为属性实现。与其他属性一样，可以用 MATLAB 的 get 函数获取这些接口中的任何一个。

例如，Microsoft Excel 组件包含多个接口，可以用不带任何参数的 get 函数，把这些接口与其他 Excel 属性列在一起。

```
h = actxserver('excel.application');
h.get
     Application: [1x1 Interface.Microsoft_Excel_9.0_
Object_Library._Application]
             Creator: 'xlCreatorCode'
               Parent: [1x1 Interface.Microsoft_Excel_9.0_
Object_Library._Application]
          ActiveCell: []
        ActiveChart: [1x50 char]
                    .
                    .
```

为了获取指定接口的句柄，在下面的例子中指定一个对象或接口的句柄 h，以及目标接口的名称 Workbooks。

```
w = h.Workbooks
w =
    Interface.Microsoft_Excel_9.0_Object_Library.Workbooks
```

2. 定制接口

下面两种服务器/客户设置方式还支持组件中可能实现的定制接口：

- MATLAB 客户程序和进程内服务器；
- MATLAB 客户程序和进程外服务器。

创建服务器以后，可以查询服务器组件，看是否实现了某些定制接口。使用 interfaces 函数返回所有可以获得的定制接口的列表。该列表在一个字符串单元数组中返回。

```
h = actxserver('mytestenv.calculator')
h =
    COM.mytestenv.calculator

customlist = h.interfaces
customlist =
    ICalc1
    ICalc2
    ICalc3
```

要获取指定定制接口的句柄，用 invoke 函数指定 actxcontrol 或 actxserver 返回的句柄，以及定制接口的名称。

```
c1 = h.invoke('ICalc1')
c1 =
   Interface.Calc_1.0_Type_Library.ICalc_Interface
```

现在可以将这个句柄与大部分 COM 客户函数一起使用，并通过选定的定制接口访问对象的属性和方法。

例如，列出通过 ICalc1 接口获得的属性，使用下面的语句。

```
c1.get
      background: 'Blue'
          height: 10
           width: 0
```

列出方法，使用下面的语句。

```
c1.invoke
   Add = double Add(handle, double, double)
   Divide = double Divide(handle, double, double)
   Multiply = double Multiply(handle, double, double)
   Subtract = double Subtract(handle, double, double)
```

下面用定制接口 c1 的 Add 和 Multiply 方法实现数字的相加和相乘。

```
sum = c1.Add(4, 7)
sum =
    11

prod = c1.Multiply(4, 7)
prod =
    28
```

6.2.6 调用 COM 对象的命令

调用 COM 对象的 MATLAB COM 函数或方法时，最简单的语法是使用点语法。指定对象名，点(.)和正在调用的函数或方法的名称。将输入参数包括在函数名后面的小括号中。在等号左侧指定输出变量。即

```
outputvalue = object.function(arg1, arg2, ...)
```

1．调用语法示例

使用本实例，首先要创建一个名为 mwsamp 的 ActiveX 控件。调用 actxcontrol 函数创建 mwsamp 控件。该函数返回句柄 h，后面将要使用它。

```
h = actxcontrol('mwsamp.mwsampctrl.2', [200 120 200 200]);
```

得到对象的句柄以后，就可以通过它调用对象的函数。下面的语句用点语法调用 addproperty 函数。调用时需要传递一个输入参数，即字符串`'Position'`。

```
h.addproperty('Position');
```

下面是其替代语法。以前诸版本都使用这种语法。

```
addproperty(h, 'Position');
```

现在 MATLAB 对这两种命令形式都支持。

2．指定属性、方法和事件名

可以下面的形式指定属性名和方法名。

```
handle.propertyname
handle.methodname
```

例如，mwsamp 对象有一个表示圆半径的 Radius 属性，一个重绘圆形的 Redraw 方法。通过键入下面的命令行获得圆的半径。

```
h.Radius
```

用下面的语句重绘圆。

```
h.Redraw
```

只要能够将属性名与其他属性的区分开，可以缩写属性名。属性名也没有大小写区分。

下面两条语句生成相同的结果。

```
x = h.Radius
x = h.r
```

方法名不能缩写。

事件名总是用函数参数中带单引号的字符串指定的。事件名必须输入完整，但没有大小写区分。下面的语句生成同样的结果。

```
h.registerevent({'MouseDown' 'mymoused'});
h.registerevent({'MOUSEdown' 'myMOUSEd'});
```

3. 隐式调用 get,set 和 invoke 函数

调用 COM 对象的 get, set 或 invoke 函数时，MATLAB 提供了更简单的语法形式。该语法形式不需要显式的函数调用。除了极少数情况外，都可以使用这种更简短的语法形式。

继续使用 mwsamp 控件。使用下面的语法，获取 Radius 属性的值并将它赋给变量 x。MATLAB 仍然调用 get 函数，但这种变短以后的语法更容易输入一些。

```
x = h.Radius
x =
    20
```

调用 set 和 invoke 函数时也可以使用这种缩写的语法。通过设置圆的新半径和调用 mwsamp 对象的 Redraw 方法放大显示圆来比较这两种调用方式。左侧的命令行显式调用 set 和 invoke 函数，右侧的命令行隐式调用它们。

```
h.set('Radius', 40);          h.Radius = 40;
h.invoke('Redraw');           h.Redraw;
```

4. 不能使用隐式语法的情况

在有些情况下，必须显式调用 get,set 和 invoke 函数，包括：

- 属性和方法非公共类型时；
- 访问带参数的属性时；
- 操作对象矢量时。

如果要访问的属性或方法不是对象类的公共属性或方法，或者不在控件或服务器的类型库中，则必须显式调用 get,set 或 invoke 函数。例如，IE 浏览器程序的 Visible 属性不是公共类型的，必须用 get 和 set 函数进行访问。

```
h = actxserver('internetexplorer.application');
% 下面的语法不合法，因为 Visible 属性不是公共的
v = h.Visible
```

```
??? No appropriate method or public field Visible for class
    COM.internetexplorer.application.

% 必须显式调用 get 函数
v = h.get('Visible')
v =
      1

% 设置非公共属性时同样要显式调用 set 函数
h.set('Visible', 1);
```

公共属性和方法可以通过下面的语句获得。

```
publicproperties = h.get
publicmethods = h.invoke
```

有些 COM 对象的属性有参数。为了获取或设置这种属性的值，必须显式调用 get 或 set 函数，如下面所示。在本例中，A1 和 B2 为指定电子表格范围的参数。

```
eActivesheetRange = e.Activesheet.get('Range', 'A1', 'B2')
eActivesheetRange =
    Interface.Microsoft_Excel_5.0_Object_Library.Range
```

如果操作对象矢量，必须显式调用 get 和 set 函数来访问属性。下面的例子创建一个包含 2 个 Microsoft Calendar 对象句柄的矢量。然后通过调用 set 函数用一次操作修改两个对象的 Day 属性。需要显式调用 get 和 set 函数。

```
h1 = actxcontrol('mscal.calendar', [0 200 250 200]);
h2 = actxcontrol('mscal.calendar', [250 200 250 200]);
H = [h1 h2];

H.set('Day', 23)
H.get('Day')
ans =
    [23]
    [23]
```

这只适合 get 和 set 函数。不能一次调用 COM 对象的多个方法，即使显式调用 invoke 函数也不能。

6.2.7 识别对象和接口

可以用表 6-1 中的函数获取与控件或服务器有关的其他信息。

表 6-1 识别对象和接口的函数

函 数	描 述
class	运行对象的类
isa	确定对象是否给定的 MATLAB 类
iscom	确定输入是否 COM 或 ActiveX 对象
isinterface	确定输入是否 COM 接口

下面的例子创建运行 Excel 的自动化服务器对象，假设对象句柄为 h，对象 Workbooks 接口的句柄为 w。

```
h = actxserver('excel.application');
w = h.Workbooks;
```

用 iscom 函数察看变量 h 是否 COM 或 ActiveX 对象的句柄。

```
h.iscom
ans =
      1
```

返回值为 1，说明 h 是 COM 或 ActiveX 对象的句柄。

下面用 isa 函数测试变量 h 是否已知类的类名。

```
h.isa('COM.excel.application')
ans =
      1
```

下面用 isinterface 函数察看变量 w 是否为 COM 接口的句柄。

```
w.isinterface
ans =
      1
```

下面用 class 函数确定变量 w 所属的类。

```
w.class
ans =
   Interface.Microsoft_Excel_9.0_Object_Library.Workbooks
```

6.2.8 调用方法

1. 操作方法的函数

使用表 6-2 中的 MATLAB 函数察看 COM 对象的方法，以及调用这些方法。

表 6-2 操作方法的函数

函　数	描　述
invoke	调用方法或显示方法和类型的列表
ismethod	确定项目是否为 COM 对象的方法
methods	列出控件或服务器的所有方法名
methodsview	列出所有方法和类型信息的 GUI 界面

使用这些函数时，输入字符串或字符串单元数组表示的事件名和事件句柄名。这些名称没有大小写区分，不能缩写。

2. 列出类或对象的方法

可以用 methodsview 函数在图形显示中，或者用 methods 函数在返回的单元数组中察看控件或服务器对象支持的方法。

methodsview 函数打开一个新窗口，窗口以一种易于阅读的方式显示指定控件或服务器对象支持的所有方法，以及几个有关的信息字段。键入下面的命令行

```
cal = actxcontrol('mscal.calendar', [0 0 400 400]);
cal.methodsview
```

生成图 6-7。

图 6-7　显示类的方法

返回 void 的方法在图中的“Retrun Type”类没有显示。

methods 函数将指定控件或服务器对象支持的所有方法的名称返回到一个单元数组中。包括可以用于对象的 MATLAB COM 函数。

在命令中包含-full 选项时，MATLAB 还给每个方法指定输入和输出参数。

```
cal.methods('-full')

Methods for class COM.mscal.calendar:

release(handle, MATLAB array)
delete(handle, MATLAB array)
MATLAB array events(handle, MATLAB array)
        .
        .
HRESULT Refresh(handle)
HRESULT Today(handle)
HRESULT AboutBox(handle)
```

调用 invoke 函数时它如果只有一个句柄参数，则返回一个类似的输出。

3．调用对象的方法

执行或调用对象的任何方法，可使用 MATLAB 的 invoke 函数，或更简单的方法名语法。invoke 函数执行对象的指定方法。可以用 invoke 函数使用下面语法中的任何一种。

```
v = invoke(handle, 'methodname', 'arg1', 'arg2', ...);
v = handle.invoke('methodname', 'arg1', 'arg2', ...);
```

下面的例子读取当前的日期，然后在一个循环中调用 NextYear 方法，将这一天提前 5 年。要获取当天的日期，键入

```
cal = actxcontrol('mscal.calendar', [0 0 400 400]);
cal.Value
ans =
    11/5/2001
```

调用 NextYear 方法将日期提前，然后确认结果。

```
for k = 1:5
    cal.NextYear;
end
```

```
cal.Value
ans =
    11/5/2006
```

与使用 invoke 函数不同，可以使用方法名来进行调用。用方法名调用的语法为：

```
v = handle.methodname('arg1', 'arg2', ...);
```

继续使用前面的例子，通过提前 5 年来返回原来的日期。

```
for k = 1:5
    cal.PreviousYear;
end

cal.Value
ans =
    11/5/2001
```

4. 指定枚举参数

使用枚举，可以用更具描述性的名称表示有些模糊的符号值。当使用的类型库报告参数为 ENUM 并且仅为 ENUM 时，MATLAB 支持将枚举作为参数传递给各方法。

下面的例子中的最后一行将枚举值'xlLocationAsObject'传递给 Microsoft Excel Chart 对象的 Location 方法。可以选择传递枚举还是传递其对应的数值。

```
e = actxserver('Excel.Application');

% 插入新的工作簿
Workbook = e.Workbooks.Add;
e.Visible = 1;
Sheets = e.ActiveWorkBook.Sheets;

% 获取活动表格的句柄
Activesheet = e.Activesheet;

% 添加图
Charts = Workbook.Charts;
Chart = Charts.Add;

% 设置图形类型为线形图
Chart.ChartType = 'xlXYScatterLines'
C1 = Chart.Location('xlLocationAsObject', 'Sheet1');
```

只处理 3 个数值型值时，记住每个值的意义还不困难。但是，如果程序需要很多这类值时，使用枚举就显得重要了。

5. 可选的输入参数

调用有可选输入参数的方法时，可以通过在对应位置上指定空数组([ ])来跳过可选参数。下面的例子中，第 2 个参数没有指定。

```
handle.methodname(arg1, [], arg3);
```

下面的例子调用 Excel 对象的 Add 方法。该方法添加新表格到 Excel 工作簿中。Add 方法有 4 个可选的输入参数，即

- Before：指定表格位置，在该表格前面添加新表格。

- After：指定表格位置，在该表格后面添加新表格。
- Count：要添加的表格的总个数。
- Type：要添加的表格的类型。

默认时在工作簿中创建 3 张表。下面的代码创建 3 张表的工作簿，然后在第 2 张表后面插入另外一张表。调用 Add 方法，只指定第 2 个参数 After。可以用[]忽略第 1 个参数。

```
%打开一个 Excel 服务器
e = actxserver('excel.application');

% 插入一个新的工作簿
e.Workbooks.Add;
e.Visible = 1;

% 获取有 3 张表的活动工作簿
eSheets = e.ActiveWorkbook.Sheets;

% 在第 2 张表后面添加一张新表
eSheet2 = eSheets.Item(2);
eNewSheet = eSheets.Add([], eSheet2);
```

6．返回多个输出参数

如果知道某服务器函数支持多个输出，可以将那些输出中的任何一个或全部返回给 MATLAB 客户程序。在等式左侧指定括弧中的输出参数。这使得 MATLAB 客户程序有权处理服务器函数返回的任何值。

下面的语法显示了 MATLAB 客户程序调用的服务器函数。函数的返回值用 retval 表示。函数的输出参数 out1,out2 等跟在后面。

```
[retval out1 out2 ...] = handle.functionname(in1, in2, ...);
```

7．出错消息中的参数编号

MATLAB 客户程序给 COM 服务器程序发送带有非法参数的命令时，服务器识别非法参数，传回与下面类似的出错消息。解释这一类消息中的参数编号时要小心。

```
PutFullMatrix(handle, 'a', 'base', 7, [5 8]);
??? Error: Type mismatch, argument 3.
```

上面的 PutFullMatrix 命令中，第 4 个参数 7 非法，因为它是标量，而不是所期望的数组数据类型。但是，出错消息把失败参数认作参数 3。

这是因为 COM 服务器只接收 MATLAB 代码中的最后 4 个参数。handle 参数只识别服务器，但没有传递给服务器。所以服务器把参数'a'看作第 1 个参数，将非法参数 7 视为第 3 个参数。

用 invoke 函数实现上面的命令时，MATLAB 客户代码中的第 5 个参数非法。但是服务器仍然把它认成第 3 个，因为前 2 个参数服务器都不识别。

```
invoke(handle, 'PutFullMatrix', 'a', 'base', 7, [5 8]);
??? Error: Type mismatch, argument 3.
```

6.2.9 对象属性

使用表 6-3 中的函数获取、设置或修改 COM 对象或接口的属性，或添加自己的定制属性。

表 6-3 操作对象属性的函数

函 数	描 述
addproperty	向 COM 对象添加一个定制属性
deleteproperty	从 COM 对象删除一个定制属性
get	列出一个或多个属性及它们的值
inspect	显示图形界面来列出和修改属性值
isprop	确定选项是否为 COM 对象的属性
propedit	显示控件的内部属性页
set	设置属性

1. 属性的值

get 函数返回 COM 对象或接口的一个或多个属性的信息。使用只有一个句柄参数的 get 函数，MATLAB 返回所有属性和属性值的列表给对象。

```
h = actxserver('excel.application');
h.get
     Application: [1x1 Interface.Microsoft_Excel_9.0_
Object_Library._Application]
         Creator: 'xlCreatorCode'
          Parent: [1x1 Interface.Microsoft_Excel_9.0_
Object_Library._Application]
      ActiveCell: []
     ActiveChart: [1x50 char]
          .
          .
```

只返回一个属性的值时，用点语法指定对象的句柄和属性名。

```
company = h.OrganizationName
company =
   The MathWorks, Inc.
```

属性名没有大小写区分，也可以缩写。例如，可以用'org'代替上面命令行中的'OrganizationName'完整属性名。

```
company = h.org
company =
   The MathWorks, Inc.
```

也可以用 get 函数函数实现，例如

```
filepath = h.get('DefaultFilePath')
filepath =
   H:\Documents
```

如果试图只用一个命令就获得多个属性，必须在一个字符串单元数组中指定属性名，并使用 get 函数。返回一个单元数组，其中的每一列对应于每个属性值。

```
C = h.get({'prop1', 'prop2', ...});
```

例如，要获得 COM 对象 h 的 DefaultFilePath 和 UserName 属性值，使用下面的语句。

```
h = actxserver('excel.application');
C = h.get({'DefaultFilePath', 'UserName'});
```

```
C{:}
ans =
    H:\Documents
ans =
    C. Coolidge
```

2. 设置属性的值

设置或修改属性值最简单的方式是使用类似下面第 2 行的赋值语句。该行语句将对象 h 的 DefaultFilePath 属性的值设置为'C:\ExcelWork'。

```
h = actxserver('excel.application');
h.DefaultFilePath = 'C:\ExcelWork';
```

也可以在不用点语法的情况下使用 set 函数来实现属性值设置。将属性名和新属性值都指定为 set 函数的输入参数。

```
h.set('DefaultFilePath', 'C:\ExcelWork');
```

用一行语句改变多个属性的值，必须使用下面的 set 函数形式。

```
h.set('prop1', 'value1', 'prop2', 'value2', ...);
```

例如，设置 COM 对象 h 的 DefaultFilePath 和 UserName 字段，使用下面的语句。

```
h = actxserver('excel.application');
h.set('DefaultFilePath', 'C:\ExcelWork', ...
    'UserName', 'H. Hoover');
```

3. 带参数的属性

某些 COM 对象的属性像方法一样可以接收参数。进行 get 或 set 函数操作时，最后获得或设置的值与传入的参数有关。

Microsoft Excel 应用程序的 Activesheet 接口作为 COM 服务器运行就是一个例子。该接口有一个 Range 属性，它实际上是另一个接口。为了获得正确的 Range 接口，必须传入指定的范围坐标。

下面代码中的第 1 行返回一个指定的 Range 接口。参数 A1 和 B2 为接口指定电子表格的某个矩形区域。

```
eActivesheetRange = e.Activesheet.get('Range', 'A1', 'B2')
eActivesheetRange =
    Interface.Microsoft_Excel_5.0_Object_Library.Range
```

获得或设置这类属性，像上面那样使用 get 或 set 函数。在属性名后面的小括号中输入参数，即

```
handle.get(propertyname, arg1, arg2, ...);
```

在某些方面，MATLAB 处理这些属性的方式与处理方法很相似，最重要的区别是处理方法需要使用 invoke 函数，而不是 get 函数来察看属性。

```
e.Activesheet.invoke
        :
    Range = handle Range(handle, Variant, Variant(Optional))
        :
```

4. 获取和设置对象矢量

通过将对象句柄放到一个矢量中然后操作该矢量，可以用 get 和 set 函数同时操作多个对象。下面的例子创建 4 个 Microsoft Calendar 对象的句柄矢量。然后通过调用 set 函数，在一次操作中修改所有对象的 Day 属性。

```
h1 = actxcontrol('mscal.calendar', [0 200 250 200]);
h2 = actxcontrol('mscal.calendar', [250 200 250 200]);
h3 = actxcontrol('mscal.calendar', [0 0 250 200]);
h4 = actxcontrol('mscal.calendar', [250 0 250 200]);
H = [h1 h2 h3 h4];

H.set('Day', 23)
H.get('Day')
ans =
    [23]
    [23]
    [23]
    [23]
```

注意：获取或设置多个对象的值，必须显式使用 get 和 set 函数。类似 H.Day 的语法只支持标量对象。

5. 使用枚举值

枚举使察看和修改属性更容易，因为每个可能的属性值都用一个字符串来表示。比如，在 Excel 应用程序中，DefaultSaveFormat 属性的默认值为 xlUnicodeText，这比一个数值值（如 57）要容易记得多。

MATLAB COM 函数 get 和 set 支持属性枚举值。用只带对象句柄参数的 set 函数察看哪些属性使用枚举类型。

```
h = actxserver('excel.application');
h.set
ans =
                    Creator: {'xlCreatorCode'}
            ConstrainNumeric: {}
         CopyObjectsWithCells: {}
                     Cursor: {4x1 cell}
                 CutCopyMode: {2x1 cell}
                          .
                          .
```

MATLAB 显示那些将枚举类型作为非空单元数组接收的属性。这些属性的设置方式用多行单元数组显示，每行对应一种设置方式。对于只有一种设置方式的属性，用单行单元数组进行显示。

显示这些属性的当前值，使用带对象句柄参数的 get 函数。

```
h.get
                    Creator: 'xlCreatorCode'
            ConstrainNumeric: 0
         CopyObjectsWithCells: 1
                     Cursor: 'xlDefault'
```

```
CutCopyMode: ''
    .
    .
```

用带属性名参数的 set 函数列出指定属性的所有枚举值。输出为字符串单元数组，其中每个字符串对应于指定属性的一种设置。

```
h.set('Cursor')
ans =
    'xlIBeam'
    'xlDefault'
    'xlNorthwestArrow'
    'xlWait'
```

将属性值设置为枚举类型，只需要简单地将枚举值赋给属性名，即

```
handle.property = 'enumeratedvalue';
```

还可以使用带属性名和枚举值的 set 函数。

```
handle.set('property', 'enumeratedvalue');
```

可以使用枚举字符串，也可以使用等价的数值值。在保证枚举值惟一性的前提下，可以缩写枚举字符串，就像下面代码中第 3 行所显示的。枚举字符串也是没有大小写区分的。

下面的例子使 Excel 电子表格窗口可见，然后从 MATLAB 客户程序改变光标。要察看光标是如何改变的，需要单击电子表格窗口。下面两种设置方式都将光标设置为沙漏类型。

```
h.Visible = 1;
h.Cursor = 'xlWait'
h.Cursor = 'xlw'            % xlWait 的缩写形式
```

读取刚刚设置的 Cursor 属性值。

```
h.Cursor
ans =
   xlWait
```

6. 使用属性探察器

MATLAB 还提供了一个图形用户界面来显示和修改属性。可以通过下面两种方式打开属性探察器。

- 从 MATLAB 命令行调用 inspect 函数。
- 在 MATLAB 工作空间浏览器中双击对象。

创建一个运行 Microsoft Excel 的服务器对象，然后将对象的 DefaultFilePath 属性设置为 'C:\ExcelWork'。

```
h = actxserver('excel.application');
h.DefaultFilePath = 'C:\ExcelWork';
```

现在调用 inspect 函数显示新窗口，该窗口中显示了服务器对象的属性。

```
h.inspect
```

打开属性探察器如图 6-8 所示。

向下滚动，直到看到刚刚改动的 DefaultFilePath 属性，其值应该为 C:\ExcelWork。

用属性探察器再改变一次 DefaultFilePath 属性值，这次改为 C:\MyWorkDirectory。设置时单击左侧的属性名，然后在右面输入新值。

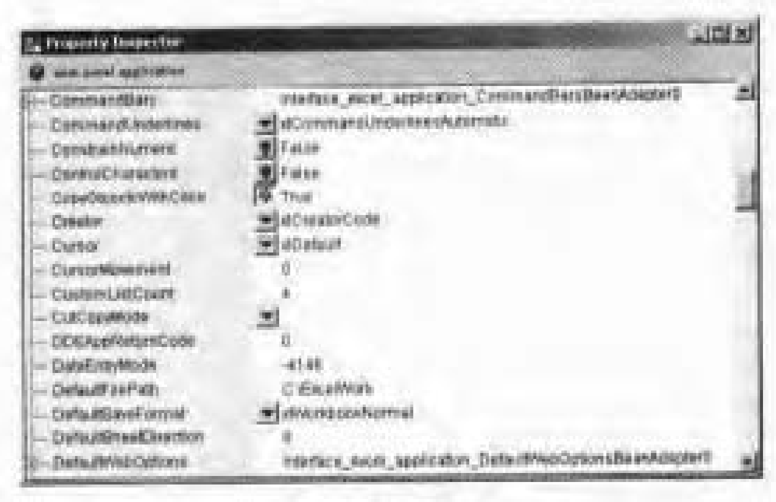


图 6-8 属性探察器

现在回到 MATLAB 命令窗口，确定 DefaultFilePath 属性值已经进行了修改。

```
h.DefaultFilePath
ans =
    C:\MyWorkDirectory
```

注意，如果在 MATLAB 命令行修改属性，必须更新"Property Inspector"窗口来察看其中的改变。通过重新调用 inspect 函数来更新"Property Inspector"窗口。

在"Property Inspector"窗口中，接受枚举值的属性用▾按钮表示。图 6-9 中显示了 Cursor 属性的 4 个枚举值。当前值用一个核选标记（√）表示。

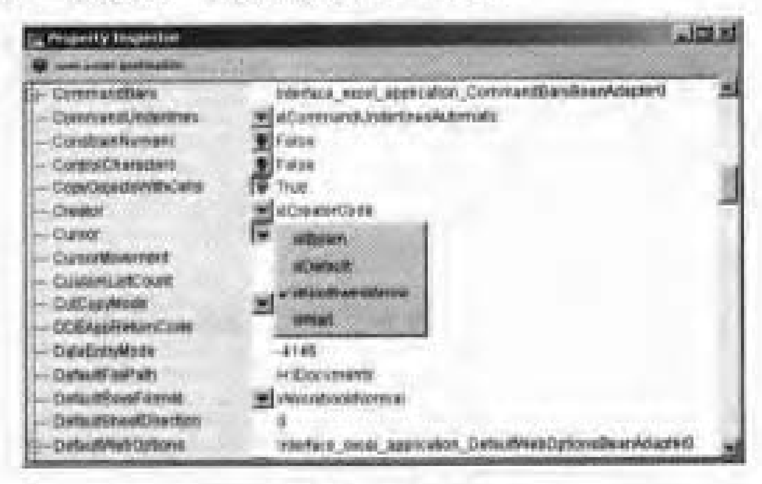


图 6-9 属性的值

用属性探察器改变属性值，只需要简单地单击▾按钮，显示该属性的选项，然后单击要选的值。

7．定制属性

使用 addproperty 函数，可以给控件添加自己的属性。下面的语法给控件 h 添加一个定制属性。

```
h.addproperty('propertyName')
```

下面创建 mwsamp2 控件，给它添加一个 Position 属性，属性值为[200 120]。

```
h = actxcontrol('mwsamp.mwsampctrl.2', [200 120 200 200]);
h.addproperty('Position');
```

```
h.Position = [200 120];
```

用 get 函数列出控件 h 的所有属性。可以发现，已经添加了新的 Position 属性。

```
h.get
ans =
       Label: 'Label'
      Radius: 20
    Position: [200 120]

h.Position
ans =
   200    120
```

从控件上删除定制属性，使用 deleteproperty 函数。

```
h.deleteproperty('propertyName')
```

例如，删除刚刚创建的 Position 属性，用 get 函数显示所有属性时它已经不存在了。

```
h.deleteproperty('Position');
h.get
       Label: 'Label'
      Radius: 20
```

6.2.10 控件和服务器事件

一般来讲，事件是用户定义的某种行为，它在服务器应用程序中发生，并且通常需要从客户应用程序作出某种反映。例如，用户在服务器界面窗口的特定位置单击鼠标可能需要客户作出某种响应动作。事件发生时，服务器告诉客户这件事情发生了。如果客户正在侦听该事件，它会通过执行事件处理程序作出响应。

MATLAB 客户可以侦听和响应 ActiveX 控件或 COM 自动化服务器发生的事件。通过注册激活的每个事件及其事件处理程序，可以选择试图侦听哪些事件。当注册后的任何事件在控件或服务器中发生时，客户得到通知，并通过执行合适的处理程序来作出响应。MATLAB 中的事件处理程序通常用 M 文件实现。

1. 用于事件处理的函数

使用表 6-4 中的 MATLAB 函数注册、注销事件，列出所有事件，或列出刚刚注册的事件。

表 6-4 用于事件处理的函数

函 数	描 述
actxcontrol	创建 COM 控件，可选地注册那些试图让客户侦听的事件
eventlisteners	返回添加给侦听程序的事件列表
events	列出所有事件，包括控件或服务器可以生成的注了册和没注册的事件
isevent	确定项目是否为 COM 对象的事件
registerevent	注册控件或服务器事件的事件处理程序
unregisterallevents	注销控件或服务器的所有事件
unregisterevent	注销控件或服务器事件的事件处理程序

使用这些函数时，用字符串或字符串单元数组输入事件名和事件处理程序名称。这些名称没有大小写区分，但不能缩写。

2. 如何准备和处理 COM 服务器的事件

下面介绍处理 COM 控件或服务器事件的基本步骤。

(1) 注册进行响应的事件

用 registerevent 函数注册那些让客户作出响应的服务器事件。有两种方法可以注册事件。

- 如果有一个处理所有服务器事件的函数，可以用下面的语法注册这个通用事件处理程序。

```
h.registerevent('handler');
```

- 如果不同类型的事件有不同的事件处理程序，可以用事件对应的处理程序注册它。使用下面的语法：

```
h.registerevent({'event1' 'handler1'; 'event2' 'handler2'; ...});
```

对于 ActiveX 控件，也可以在使用 actxcontrol 函数创建控件时注册事件（有下面两种方法）。该函数将句柄 h 返回给新创建的控件对象。

- 使用下面的语法注册对所有事件都作出响应的通用事件处理程序。

```
h = actxcontrol('progid', position, figure, 'handler');
```

- 使用下面的语法注册适用于某种事件类型的处理函数。

```
h = actxcontrol('progid', position, figure, ...
    {'event1' 'handler1'; 'event2' 'handler2'; ...});
```

MATLAB 客户只侦听那些已经注册的事件。

(2) 识别所有事件和注了册的事件

使用 events 函数列出控件或服务器可以作出响应的所有事件。该函数返回一个结构数组，其中结构的每个字段是事件处理程序的名称，字段值包含处理程序的函数形式。调用句柄为 h 的对象的 events 函数，键入

```
S=h.events
```

eventlisteners 函数只列出当前注册的那些事件。该函数返回一个单元数组，其中的每一行表示一个注册了的事件及其事件处理程序的名称。调用句柄为 h 的对象的 eventlisteners 函数，键入

```
C=h.eventlisteners
```

(3) 事件发生时作出响应

不管什么时候控件或服务器触发了客户正在侦听的事件，客户都会通过调用已经注册的一个或多个事件处理程序来响应事件。可以在处理事件的 M 文件程序中实现这些函数。

(4) 注销不再需要侦听的事件

如果已经注册了一些事件，但现在想让客户程序忽略它，可以用 unregisterevent 和 unregisterallevents 函数注销它们。

- 对于句柄为 h 的服务器，注销作为通用事件处理程序注册的所有事件，使用下面的语法：

```
h.unregisterevent('handler');
```

- 注销作为单独事件处理程序注册的事件 eventN，使用下面的语法：

```
h.unregisterevent({'event1' 'handler1'; 'eventN' 'handlerN'});
```

- 注销全部事件，使用下面的语法：

```
h.unregisterallevents;
```

3. 响应 ActiveX 控件的事件

本例演示如何处理 ActiveX 控件触发的事件。使用 MATLAB 自带的 mwsamp2 控件。安装 MATLAB 时就定义了 mwsamp2 的事件处理程序。

- 注册控件事件：actxcontrol 函数不仅仅创建控件对象，还可以用于注册指定的事件。下面的代码注册 2 个事件(Click 和 MouseDown)，以及 2 个处理函数(myclick 和 mymoused)。

```
f = figure('position', [100 200 200 200]);
h = actxcontrol('mwsamp.mwsampctrl.2', [0 0 200 200], f, ...
    {'Click' 'myclick'; 'MouseDown' 'mymoused'});
```

如果后面还要注册其他事件，使用 registerevent 函数。

```
h.registerevent({'DblClick' 'my2click'});
```

继续本实例以前注销 DblClick 事件。

```
h.unregisterevent({'DblClick' 'my2click'});
```

- 列出控件的事件：这时，应该只注册了 Click 和 MouseDown 事件。要察看控件可以触发的所有事件，使用 events 函数。该函数返回一个结构数组，其中结构的每个字段为事件处理程序的名称，字段值包含处理函数的形式。

要列出所有事件，使用下面的语句：

```
S = h.events
S =
          Click: 'void Click()'
       DblClick: 'void DblClick()'
      MouseDown: 'void MouseDown(int16 Button, int16 Shift,
                  Variant x, Variant y)'
     Event_Args: [1x101 char]

S.Event_Args
ans =
   void Event_Args(int16 typeshort, int32 typelong,
        double typedouble, string typestring, bool typebool)
```

如果只列出那些当前用控件注册了的事件，使用 eventlisteners 函数。该函数返回一个单元数组，其中的每一行表示一个注册了的事件及其事件处理程序的名称。

使用 eventlisteners 函数列出注了册的事件的名称和它们的处理函数。

```
h.eventlisteners
ans =
    'click'         'myclick'
    'mousedown'     'mymoused'
```

- 响应控件的事件：MATLAB 创建 mwsamp2 控件时，还显示中间有标签和圆的图形窗口。如果单击该窗口中的不同位置，可以在 MATLAB 命令窗口中看到鼠标位置的 X 坐标和 Y 坐标。

每次按下鼠标键时，都会触发 MouseDown 事件，调用 mymoused 函数。该函数将点的位置值输出到 MATLAB 命令窗口中。

```
The X position is:
ans =
    [122]
The Y position is:
```

```
ans =
    [63]
```

还可以查看响应 Click 事件时报告的消息。

```
Single click function
```

双击鼠标没有任何不同，因为 DblClick 事件已经注销了。

● 注销控件的事件：注销事件时，客户不再侦听该事件的发生。事件触发时，客户不会作出响应。如果注销 MouseDown 事件，你会发现，在窗口上单击时 MATLAB 不再报告 *X* 和 *Y* 的位置。

```
h.unregisterevent({'MouseDown' 'mymoused'});
```

现在，注册 DblClick 事件，将它与处理函数 my2click 连接。

```
h.registerevent({'DblClick', 'my2click'});
```

如果再次调用 eventlisteners 函数，会发现现在注了册的事件为 Click 和 DblClick。

```
h.eventlisteners
ans =
    'click'        'myclick'
    'dblclick'     'my2click'
```

双击鼠标键时，MATLAB 会在 MATLAB 命令窗口中显示下面消息，先后响应 Click 和 DblClick 事件。

```
Single click function
Double click function
```

注销控件所有事件的一个简单方法是使用 unregisterallevents 函数。没有事件注册时，eventlisteners 函数返回一个空的单元数组。

```
h.unregisterallevents
h.eventlisteners
ans =
    {}
```

现在在控件窗口中单击鼠标不发生任何事情，因为没有任何事件被触发。

如果有注册为通用事件处理程序的事件，例如用在下面例子中的 sampev.m，可以用 unregisterevent 函数在一次操作中注销所有这些事件。下面的例子用通用处理程序 sampev 在服务器中注册所有事件。现在 MATLAB 通过执行 sampev 从该服务器中处理所有类型的事件。

```
h.registerevent('sampev');
```

通过列出所有事件侦听程序来确认刚才的注册。

```
h.eventlisteners
ans =
    'click'        'sampev'
    'dblclick'     'sampev'
    'mousedown'    'sampev'
```

现在注销所有使用 sampev 事件处理程序的事件。

```
h.unregisterevent('sampev');
h.eventlisteners
ans =
    {}
```

4. 从自动化服务器响应事件示例

下面的例子演示如何处理自动化服务器触发的事件。它创建一个运行 IE 的服务器，为所有事件注册一个通用处理程序，然后通过用 IE 程序浏览网址来触发事件。

● 注册服务器事件：下面的例子注册自动化服务器触发的所有事件，与上面的例子不同，注册同一个处理程序 serverevents 来响应所有类型的事件。因为本例不是 MATLAB 自带的，所以必须自己创建事件处理程序。创建文件 serverevents.m，代码如下所示。

```
function serverevents(varargin)
% 显示引入的事件名
eventname = varargin{end}

% 显示引入的事件参数
eventargs = varargin{end-1}
```

然后使用下面的命令创建自动化服务器应用程序，并从服务器中为所有事件注册处理程序。

```
% 创建一个运行 IE 的服务器
h = actxserver('internetexplorer.application');

% 使服务器应用程序可见
h.set('Visible', 1);

%从服务器用通用事件处理程序注册所有事件
h.registerevent('serverevents');
```

● 列出服务器的事件：用 events 函数列出控件或服务器可以响应的所有事件，用 eventlisteners 函数只列出当前注册的那些事件。

```
h = actxserver('internetexplorer.application');
h.Visible = 1;

h.events
    StatusTextChange = void StatnsTextChange(string Text)
    ProgressChange = void ProgressChange(int32 Progress,
        int32 ProgressMax)
    CommandStateChange = void CommandStateChange(int32 Command,
        bool Enable)
            :
            :

% 这时没有注册事件，所以侦听程序返回空的单元数组
h.eventlisteners
ans =
      {}

h.registerevent('serverevents');
h.eventlisteners
ans =
     'statustextchange'         'serverevents'
     'progresschange'           'serverevents'
     'commandstatechange'       'serverevents'
              :                        :
              :                        :
```

● 响应服务器事件：现在全部事件都注册了。如果触发任何事件，将执行 serverevents.m 定义的通用处理程序来处理该事件。用 IE 浏览器程序浏览网站，或者在 MATLAB 命令行中键入下面的命令。

```
h.Navigate2('http://www.mathworks.com');
```

应该看到显示在客户窗口中的一长串事件。

● 注销服务器事件：使用 unregisterevent 函数注销指定的事件。用字符串单元数组指定要注销的每个事件及其处理函数。

```
h.unregisterevent({'event1', 'handler1'; ...
    'event2', 'handler2', ...});
```

如果事件用通用处理程序注册，就像本例，则需要给要注销的每个事件指定通用处理程序的名称。

```
h.unregisterevent({'event1', 'commonhandler'; ...
    'event2', 'commonhandler', ...});
```

继续本例，注销 progresschange 和 commandstatechange 事件。

```
h.unregisterevent({'progresschange', 'serverevents'; ...
    'commandstatechange', 'serverevents'});
```

使用 unregisterallevents 函数可以注销对象的所有事件。下面两个命令注销所有事件，然后只注册一个单独的事件。

```
h.unregisterallevents;
h.registerevent({'TitleChange', 'serverevents'});
```

如果现在用 IE 进行浏览，MATLAB 会只响应 TitleChange 事件。

● 关闭应用程序：程序不再使用时就应该关闭。可以通过键入下面的语句来注销所有事件并关闭应用程序。

```
h.unregisterallevents;
h.Quit;
h.delete;
```

6.2.11 编写事件处理程序

1. 事件处理概述

当控件或服务器试图通知其客户有事发生时，触发事件。例如，用户在控件界面窗口中的某处单击时，许多控件都触发事件。在 MATLAB 中，可以创建和注册自己的 M 文件函数，使得事件发生时可以作出响应。这些函数就是事件处理程序。可以创建一个处理程序处理所有事件，也可以给每类事件安排一个单独的处理程序。

对于控件，可以在创建控件时或者以后任何时候注册处理程序。注册处理程序可分别使用 actxcontrol 函数和 registerevent 函数。对于服务器，必须用 registerevent 函数注册那些试图让客户侦听的事件。在参数列表中指定事件处理程序，该参数可以是回调函数的名称，也可以是由指定事件和它们各自的事件处理程序组成的单元数组。

```
h = actxcontrol (progid, position, handle, ...
callback | {event1 eventhandler1; event2 eventhandler2; ...})
```

指定单一回调过程时，MATLAB 用该过程注册所有事件。发生任何事件时，MATLAB 执

行通用的 callback 过程。

可以用 events 函数列出 COM 对象识别的所有事件。例如，使用下面的语句列出 mwsamp2 控件的所有事件。

```
f = figure ('position', [100 200 200 200]);
h = actxcontrol ('mwsamp.mwsampctrl.2', [0 0 200 200], f);
h.events
    Click = void Click()
    DblClick = void DblClick()
    MouseDown = void MouseDown(int16 Button, int16 Shift,
        Variant x, Variant y)
```

2．传递给事件处理程序的参数

触发注了册的事件时，MATLAB 如表 6-5 中所示将信息从事件传递给它的处理程序。

表 6-5　MATLAB 传递的参数

第几个参数	内　容	格　式
1	对象名称	MATLAB COM 类
2	事件 ID	double
3	事件参数列表的开始	与控件传递的相同
倒数第 3	事件参数列表的末尾	与控件传递的相同
倒数第 2	事件结构	结构
最后一个	事件名	字符数组

编写事件处理程序时，用事件名参数识别事件的来源，从事件参数列表中获取控件传递的参数。所有事件处理程序必须接受多个参数。

```
function event (varargin)
if (varargin{end}) == 'MouseDown')        % 检查事件名
    x_pos = varargin{5};                  % 读取第 5 个事件参数
    y_pos = varargin{6};                  % 读取第 6 个事件参数
end
```

3．事件结构

MATLAB 传递的倒数第 2 个参数是事件结构，它具有表 6-6 所示的字段。

表 6-6　事件结构的字段

字 段 名	描　述	格　式
Type	事件名	字符数组
Source	控件名	MATLAB COM 类
EventID	事件标识符	double
Event Arg Name 1	事件参数值 1	与控件传递的相同
Event Arg Name 2	事件参数值 2	与控件传递的相同
…	事件参数 N	与控件传递的相同

例如，触发 mwsamp2 控件的 MouseDown 事件时，MATLAB 将这个事件结构传递给注了册的事件处理程序。

```
        Type: 'MouseDown'
      Source: [1x1 COM.mwsamp.mwsampctrl.2]
     EventID: -605
      Button: 1
       Shift: 0
           x: 27
           y: 24
```

4．事件处理程序示例

下面指定单一回调 sampev：

```
f = figure('position', [100 200 200 200]);
h = actxcontrol('mwsamp.mwsampctrl.2', [0 0 200 200], ...
    gcf, 'sampev')
h =
    COM.mwsamp.mwsampctrl.2
```

或者用单元数组格式指定几个事件。

```
h = actxcontrol('mwsamp.mwsampctrl.2', [0 0 200 200], f, ...
    {'Click' 'myclick'; 'DblClick' 'my2click'; ...
    'MouseDown' 'mymoused'});
```

事件处理程序 myclick.m,my2click.m 和 mymoused.m 为：

```
function myclick(varargin)
disp('Single click function')

function my2click(varargin)
disp('Double click function')

function mymoused(varargin)
disp('You have reached the mouse down function')
disp('The X position is: ')
double(varargin{6})
disp('The Y position is: ')
double(varargin{7})
```

另外，可以用单元数组对将同一个事件处理程序用于所有试图监测的事件。使用这种方法时，响应时间会比使用回调类型的快。例如：

```
f = figure('position', [100 200 200 200]);
h = actxcontrol('mwsamp.mwsampctrl.2', ...
[0 0 200 200], f, {'Click' 'allevents'; ...
'DblClick' 'allevents'; 'MouseDown' 'allevents'})
```

其中，allevents.m 的代码为：

```
function allevents(varargin)
if (strcmp(varargin{3}.Type, 'Click'))
    disp ('Single Click Event Fired')
elseif (strcmp(varargin{3}.Type, 'DblClick'))
    disp ('Double Click Event Fired')
elseif (stromp(varargin{3}.Type, 'MouseDown'))
    disp ('Mousedown Event Fired')
end
```

5. 用 M 文件子函数编写事件处理程序

其实用不着给每个事件处理程序编写一个单独的 M 文件，可以用 M 文件子函数将这些处理函数中的某些或全部放到一个单一的 M 文件中。

下面的例子显示了文件 mycallbacks.m 中用子函数实现的 3 个事件处理程序，即 myclick,my2click 和 mymoused。调用 str2func 函数将输入的字符串转换为函数句柄。

```
function a = mycallbacks(str)
a = str2func(str);

function myclick(varargin)
disp('Single click function')

function my2click(varargin)
disp('Double click function')

function mymoused(varargin)
disp('You have reached the mouse down function')
disp('The X position is: ')
double(varargin{6})
disp('The Y position is: ')
double(varargin{7})
```

注册这些事件中的一个，只需要调用 mycallbacks 函数，并传递事件处理程序的名称。

```
h = actxcontrol('mwsamp.mwsampctrl.2', [0 0 200 200], ...
    gcf, 'sampev')
h.registerevent('Click', mycallbacks('myclick'));
```

6.2.12 保存工作

使用表 6-7 中的 MATLAB 函数保存和恢复 COM 控件对象的状态。

表 6-7 实现保存和恢复的函数

函 数	描 述
load	从文件中载入和初始化 COM 控件对象
save	将 COM 控件对象写入和序列化到文件

用 save 函数将 COM 控件的当前状态保存到文件。下面的例子创建一个 mwsamp2 控件，并将它的初始状态保存到文件 mwsample 中。

```
f = figure('position', [100 200 200 200]);
h = actxcontrol('mwsamp.mwsampctrl.2', [0 0 200 200], f);
h.save('mwsample')
```

现在，改变标签和圆的半径。

```
h.Label = 'Circle';
h.Radius = 50;
h.Redraw;
```

使用 load 函数，可以将控件恢复到初始状态。

```
h.load('mwsample');
h.get
```

```
ans =
    Label: 'Label'
   Radius: 20
```

注意，COM save 和 load 函数目前只支持控件。

6.2.13 释放 COM 接口和对象

使用表 6-8 中的 MATLAB 函数释放或删除 COM 对象或接口。

表 6-8 释放和删除 COM 对象或接口的函数

函　数	描　述
delete	删除 COM 对象或接口
release	释放 COM 对象或接口

当接口不需要时，用 release 函数释放接口，收回它使用的内存。当整个控件或服务器都不需要了时，用 delete 函数删除它。也可以将 delete 函数用于任何合法的接口。对象的所有接口自动释放，并且控件或服务器本身也被删除。

当包含控件的图形窗口被删除或关闭时，MATLAB 自动释放该控件的所有接口。MATLAB 关闭时，它还自动释放自动化服务器的所有句柄。

6.2.14 识别对象

使用表 6-9 中的函数可获取与 COM 对象有关的信息。

表 6-9 获取 COM 对象信息的函数

函　数	描　述
class	返回 COM 对象的类
isa	删除给定类的 COM 对象
isevent	确定项目是否为 COM 对象的事件
ismethod	确定项目是否为 COM 对象的方法
isprop	确定项目是否为 COM 对象的属性

创建一个运行 Excel 的自动化服务器 COM 对象 h，以及对象的 Workbooks 接口 w。

```
h = actxserver('excel.application');
w = h.Workbooks;
```

用 class 函数查找变量 w 所属的类。

```
w.class
ans =
   Interface.Microsoft_Excel_9.0_Object_Library.Workbooks
```

用 isa 函数测试变量是否属于已知的类。

```
h.isa('COM.excel.application')
ans =
     1
```

用 isprop 函数检查 UsableWidth 是否为对象 h 的属性。

```
h.isprop('UsableWidth')
ans =
     1
```

用 ismethod 函数检查 SaveWorkspace 是否为对象 h 的方法。方法名是有大小写区分的，不能缩写。

```
h.ismethod('SaveWorkspace')
ans =
     1
```

创建 mwsamp2 控件，用 isevent 函数检查 DblClick 是否为该控件可以识别的事件。

```
f = figure ('position', [100 200 200 200]);
h = actxcontrol ('mwsamp.mwsampctrl.2', [0 0 200 200], f);

h.isevent('DblClick')
ans =
     1
```

6.2.15 MATLAB 作为自动化客户示例

这里提供一个将 MATLAB 用作自动化客户的实例。

1. MATLAB 控件实例

MATLAB 自带了一个简单的 COM 控件，可以实现画圆和显示文本，用户在控件上单击或双击时，触发事件。通过运行目录 winfun\comcli 下的 mwsamp.m 文件，或键入下面的命令行来创建控件。

```
h = actxcontrol('mwsamp.mwsampctrl.2', [0 0 300 300]);
```

这个控件保存在 MATLAB 的 bin 目录下，与控件的类型库放在一起。类型库是 COM 工具用于解释控件功能的二进制文件。

2. 将 MATLAB 用作自动化客户

本例将 MATLAB 用作自动化客户，将 Excel 用作服务器。

```
% MATLAB 自动化客户示例
%
% 打开 Excel，添加工作簿，改变活动表格
% 获取/放置数组，保存

% 首先，打开一个 Excel 服务器
e = actxserver('excel.application');

% 插入一个新的工作簿
eWorkbook = e.Workbooks.Add;
e.Visible = 1;

% 使第 2 个表格成为活动的
eSheets = e.ActiveWorkbook.Sheets;

eSheet2 = eSheets.get('Item', 2);
eSheet2.Activate;

% 将 MATLAB 数组放到 Excel 中
A = [1 2; 3 4];
```

```
eActivesheetRange = e.Activesheet.get('Range', 'A1:B2');
eActivesheetRange.Value = A;

% 回到一个范围。它将是单元数组，因为单元类型可以包含不同类型的数据
eRange = e.Activesheet.get('Range', 'A1:B2');
B = eRange.Value;

% 转换为 double 型矩阵。单元数组必须只包含标量
B = reshape([B{:}], size(B));

% 现在保存工作簿
eWorkbook.SaveAs('myfile.xls');

%不保存工作簿，去掉下面语句行的注释符号
% eWorkbook.Saved = 1;
% eWorkbook.Close;

% 退出 Excel，删除服务器
% e.Quit;
% e.delete;
```

注意，要确保关闭了添加到 Excel 中的所有工作簿。这样可以防止潜在的内存泄露问题。

6.3 其他 COM 客户信息

6.3.1 使用 COM 集合

COM 集合是支持成组相关 COM 对象的一种方式。集合本身是一个特殊接口。它有一个 Count 属性和一个 Item 方法。Count 属性包含集合中项目的个数，Item 方法指定对集合中的哪一个项目进行处理，所以它实际上是一种索引。索引的数据类型可以是任何适用于特定集合的数据类型。通常，Item 方法的返回值本身就是一个接口。就像其他接口一样，使用完以后需要进行释放。

本例遍历集合的成员。每个成员本身就是一个接口（称为 plot，用 MATLAB COM 对象表示时为 hPlot）。在特定情况下，会遍历 Plot 接口集合，调用每个接口的 Redraw 方法，然后释放每个接口：

```
hCollection = hControl.Plots;
for i = 1:hCollection.Count
    hPlot = hCollection.invoke('Item', i);
    hPlot.Redraw;
    hPlot.release;
end;
hCollection.release;
```

6.3.2 转换数据

因为 COM 定义了很多不同的数据格式和类型，所以需要知道 MATLAB 如何将数据从 COM 对象转换为 MATLAB 工作空间中的变量。当提取属性值或者从方法返回值时，必须转

换 COM 对象的数据。

表 6-10 显示了 COM 数据类型是如何转换为 MATLAB 空间中的变量的。

表 6-10　COM 数据类型与 MATLAB 数据类型之间的对应关系

COM 数据类型	MATLAB 数据类型	COM 数据类型	MATLAB 数据类型
String	MATLAB String	Variant	单元数组
File Time		Variant 数组	
Error		IDispatch *	COM 对象
Decimal Date		Empty	不能转换
Currency	Scalar Double	Unknown	
Hresult		Void	
Int/Unsigned(2,4,8)		Ptr	
Bool		Carray	
Real(Single/Double)		Userdefined	
Null	NaN	Blob	
Currency 数组	Double 型矩阵	Stream	
Hresult 数组		Storage	
Int/Unsigned(2,4,8)数组		Streamed 对象	
Bool 数组		Stored 对象	
Real(Single/Double)数组		Blob 对象	
		CF	

6.3.3　将 MATLAB 用作 DCOM 客户程序

分布式组件对象模型（DCOM）是一项协议，它允许客户通过网络使用远程 COM 对象。另外，如果 MATLAB 所在的操作系统运行进行 DCOM 操作，则 MATLA 可以作为 DCOM 客户程序与远程自动化服务器一起使用。

注意，如果将 MATLAB 用作远程 DCOM 服务器，所有 MATLAB 窗口将出现在远程机器上。

6.3.4　MATLAB　COM 支持的局限性

MATLAB COM 支持有下面几点局限性：

- MATLAB 只支持索引集合；
- 设计时 COM 控件不在图形窗口上显示出来；
- MATLAB 只支持控件事件，不支持服务器事件。

6.4　MATLAB 自动化服务器支持

运行于 Windows 系统的 MATLAB 支持 COM 自动化服务器功能。自动化是一种 COM 协议，它允许一个应用程序或组件（控制者）控制另一个应用程序或组件（服务器）。这样，MATLAB 服务器可以被任何可作为自动化控制者的 Windows 程序控制。可作为自动化控制者的应用程序包括 Excel、Access、Project 和很多 Visual Basic 和 Visual C++程序。

下面介绍如何创建和连接运行 MATLAB 的自动化服务器，如何从 MATLAB M 文件或 Visual Basic 客户程序调用服务器中的函数，以及如何使用影响服务器的属性。

注意，如果想用 C 或 Fortran 生成客户程序，建议使用 MATLAB 引擎工具，而不是自动化服务器。

6.4.1 创建自动化服务器

究竟如何创建自动化服务器，与正在使用的控制程序有关。查询控制程序的文档应该可以找到这方面的信息。所有控制程序都需要一个程序标识符(ProgID)来识别服务器。MATLAB 的 ProgID 为 matlab.application。

如果控制程序为 MATLAB，可以用 MATLAB 的 actxserver 函数创建自动化服务器。

```
h = actxserver('matlab.application')
h =
   COM.matlab.application
```

通常，当控制程序第一次与服务器建立连接时，Windows 就会自动建立自动化服务器。也可以手工创建服务器。

1. 共享模式和独占模式

启动 MATLAB 自动化服务器时可以有 2 种模式，即共享模式和独占模式。使用共享模式时，一个或多个客户程序与同一个 MATLAB 服务器相连接。服务器在所有客户程序间共享。使用独占模式时，每个客户程序创建它自己专用的 MATLAB 服务器。

如果将 matlab.application 用作 ProgID，MATLAB 会创建一个共享模式的服务器。

2. 启动目录

自动化服务器在 MATLAB 根目录的\bin\win32 子目录下启动。如果这不是你常用的 MATLAB 启动目录，则服务器进程启动 MATLAB 时，新创建的服务器将无权访问该目录中的文件，也不会运行 MATLAB 启动文件(startup.m)。

为了使服务器有权访问启动目录中的文件，需要让服务器用 cd 函数将工作目录设置为启动目录，然后用 addpath 函数将启动目录添加到服务器的 MATLAB 路径；或者在引用这些文件时将路径名包含到启动目录中。

6.4.2 连接已经存在的服务器

程序完成操作任务时，并不总是需要创建一个新的 MATLAB 服务器实例。客户可以用与 Visual Basic 的 GetObject 命令类似的命令连接已经存在的 MATLAB 自动化服务器。

这里显示的 Visual Basic 命令返回 MATLAB 服务器应用程序的句柄 h。

```
h = GetObject(, "matlab.application")
```

下面的 Visual Basic 代码连接一个已经存在的 MATLAB 服务器，然后在服务器中执行一个绘图命令。如果没有已经存在的 MATLAB 服务器，可以先创建一个。然后，在 Visual Basic 程序中输入下面的代码，连接服务器并让 MATLAB 画一个简单的图。

```
Dim h As Object
Set h = GetObject(, "matlab.application")

' 句柄 h 现在应该是合法的，
' 通过调用 Execute 方法来测试
h.Execute ("plot([0 18], [7 23])")
```

6.4.3 自动化服务器函数

MATLAB 提供了很多函数，使得 MATLAB 或 Visual Basic 编写的自动化控制程序可以操作 MATLAB 服务器中的数据。这些函数如表 6-11 中所示。

1．在 MATLAB 服务器中执行命令

客户程序可以用表 6-11 中的函数在服务器中执行 MATLAB 命令。

表 6-11 执行 MATLAB 命令的函数

函 数	描 述
Execute	在服务器中执行 MATLAB 命令
Feval	在服务器中处理 MATLAB 函数

命令可以用单独一条字符串表示时，使用 Execute 函数，例如

```
h = actxserver('matlab.application');

h.PutWorkspaceData('A', 'base', rand(6));
h.Execute('A(4:6,:) = [];');
B = h.GetWorkspaceData('A', 'base')
B =
    0.6208    0.2344    0.6273    0.3716    0.7764    0.7036
    0.7313    0.5488    0.6991    0.4253    0.4893    0.4850
    0.1939    0.9316    0.3972    0.5947    0.1859    0.1146
```

命令不能用单独一条字符串表示时，使用 Feval 函数。下面的例子中，命令中包含不能在服务器中定义的变量 rows,cols 和 pages。继续上面的实例。

```
rows = 3;    cols = 3;    pages = 2;
h.Feval('reshape', 1, {'A=', rows, cols, pages});

B = h.GetWorkspaceData('A', 'base')
B(:, :, 1) =
    0.6208    0.2344    0.6273
    0.7313    0.5488    0.6991
    0.1939    0.9316    0.3972
B(:, :, 2) =
    0.3716    0.7764    0.7036
    0.4253    0.4893    0.4850
    0.5947    0.1859    0.1146
```

2．与服务器交换数据

MATLAB 提供了几个函数来实现与 MATLAB 服务器交换数据，如表 6-12 中所示。调用这些函数时，需要传递要读写的变量的名称，以及包含数据的工作空间的名称。

表 6-12 与服务器交换数据的函数

函 数	描 述
GetCharArray	从服务器获取字符数组
GetFullMatrix	从服务器获取矩阵
GetWorkspaceData	从服务器获取任何类型的数据

续表

函　数	描　述
PutCharArray	将字符数组保存到服务器中
PutFullMatrix	将矩阵保存到服务器中
PutWorkspaceData	将任何类型的数据保存到服务器中

Get/PutCharArray 函数实现从 MATLAB 服务器读取字符串值，以及写字符串值到 MATLAB 服务器。

Get/PutCharArray 函数将数据作为 SAFEARRAY 数据类型进行传递。可以将这些函数与任何支持 SAFEARRAY 类型的客户程序一起使用。

Get/PutWorkspaceData 函数将数据作为 variant 类型传递。将它们与支持 variant 类型的客户程序一起使用。这些函数对于 VBScript 客户程序很有用，因为 VBScript 程序不支持 SAFEARRAY 数据类型。

在 MATLAB 服务器基本工作空间中将一个字符串写入变量 str，然后将它读回到客户程序。

```
h = actxserver('matlab.application');
h.PutCharArray('str', 'base', ...
    'He jests at scars that never felt a wound.');

S = h.GetCharArray('str', 'base')
S =
  He jests at scars that never felt a wound.
```

3. 控制服务器窗口

使用表 6-13 中的函数，可以使服务器窗口可见或最小化。

表 6-13　控制服务器窗口的函数

函　数	描　述
MaximizeCommandWindow	在窗口桌面上显示服务器窗口
MinimizeCommandWindow	最小化服务器窗口的大小

创建运行 MATLAB 的 COM 服务器，并使之最小化。

```
h = actxserver('matlab.application');
h.MinimizeCommandWindow;
```

4. 终止服务器进程

用 MATLAB 服务器完成任务以后，用 Quit 函数退出 MATLAB 并终止服务器进程。

终止服务器进程，只需输入下面的命令。

```
h.Quit;
```

5. 客户详细信息

对于 MATLAB 客户，查看控制程序可以使用的所有函数及其语法，启动一个 MATLAB 自动化服务器，然后使用只带 handle 参数的 invoke 函数，即

```
handle = actxserver('matlab.application');
handle.invoke
```

对于 Visual Basic 客户，服务器函数的参数和返回值的数据类型用自动化数据类型表示，

它们是自动化协议定义的与语言无关的类型。例如，BSTR 是定义为自动化类型的宽字符字符串类型，与 Visual Basic 保存字符串的数据格式相同。任何服从 COM 协议的控制程序都应该支持这些数据类型，即使声明和操作细节有所区别。

6.4.4 MATLAB 自动化属性

通过设置 Visible 属性，可以选择是否让服务器程序可见。可见时，服务器窗口显示在桌面上，使用户可以与服务器程序进行交互。在某些时候，例如调试时，这会很有用。不可见时，不显示服务器窗口。这样可能会使界面更干净，但也不利于与服务器程序进行交互。

默认时，Visible 属性是可用的，或设置为 1。

```
h = actxserver('matlab.application');
h.Visible
ans =
     1
```

通过设置为 0（不可见）或 1（可见）来改变 Visible 属性的设置。下面的命令将服务器应用程序窗口从桌面上删除。

```
h.Visible = 0;
h.Visible
ans =
     0
```

6.5 其他自动化服务器信息

6.5.1 手工创建服务器

控制者应用程序第一次建立与服务器的连接时，Windows 自动创建自动化服务器。也可以选择在启动任何客户进程以前手工创建服务器。

手工创建服务器，需要在 MATLAB 启动命令中使用/Automation 开关。按照下面的步骤进行。

（1）在桌面上右击图标，在弹出式菜单中单击“属性”选项，打开“属性”对话框，默认时打开的是“快捷方式”选项卡。

（2）在“目标”文本框中，在 matlab.exe 的目标路径后面添加/Automation。

6.5.2 指定共享或独占服务器

可以用共享模式或独占模式启动 MATLAB 自动化服务器。独占模式的服务器只供一个客户程序使用，共享模式的服务器可以由多个客户共同使用。使用哪种模式，由启动 MATLA 时客户使用的程序标识符确定。

1．启动共享式服务器

ProgID 值 matlab.application 指定默认模式，即共享模式。还可以使用区分版本的 ProgID，即 matlab.application.N。其中 N 等于正在运行的 MATLAB 的主版本，例如，对于 MATLAB 6.5，N=6。

MATLAB 作为共享模式的服务器启动以后，所有客户程序都会通过使用与已经在运行的

MATLAB 实例相连接的共享服务器 ProgID 来请求连接到 MATLAB。换句话说，绝对不可能有一个以上共享服务器的实例同时运行，因为它被所有使用 ProgID 的客户共享。

2. 启动独占式服务器

指定独占式服务器，使用 ProgID matlab.application.single，或者指定版本的 ProgID matlab.application.single.N。

每个用独占式 ProgID 请求与 MATLAB 建立连接的客户都创建一个单独的 MATLAB 实例，并且该服务器不会被任何其他客户共享。所以，可以有多个独占式服务器的实例同时运行，因为独占式服务器不被多个客户共享。

6.5.3 将 MATLAB 用作 DCOM 服务器

DCOM 是一项协议，它允许通过网络建立 COM 连接。如果正在使用支持 DCOM 的 Windows 版本，以及一个支持 DCOM 的控制程序，可以使用控制程序在远程机器上启动 MATLAB 服务器。

要实现这一点，DCOM 必须正确注册，并且在每台用作客户或服务器的机器上安装 MATLAB。

第 7 章　MATLAB 与 C 接口

7.1　MATLAB 与 C 接口概述

MATLAB 将数值分析、矩阵计算、信号处理和图形显示结合在一起，包含大量高度集成的函数可供调用，命令语句功能十分强大，为科学研究、工程设计及众多学科领域提供了一种简洁、高效的编程工具。但是 MATLAB 使用的是解释性语言，大大限制了它的执行速度；源代码的公开不利于算法和数据的保密；局限于 MATLAB 运行环境而不能用于开发商用软件；GUI 功能差，所提供的控件和事件实在有限。因此，如果能够实现 MATLAB 与 VC 或者 BC 或者 C++ BUILDER 等可视化设计语言的交互，提高速度，美化界面，使程序更符合 Windows 的规范，同时又利用 MATLAB 的强大功能，对任何人来说都是很有意义的。本章主要介绍 MATLAB 与 C/C++接口方面的知识。

MATLAB 与 C/C++接口主要有以下三种方式。

1. MEX 文件

在 MATLAB 中可调用的 C 或 Fortran 语言程序称为 MEX 文件。MATLAB 可以直接把 MEX 文件视为它的内建函数进行调用。MEX 文件是动态链接的子例程，MATLAB 解释器可以自动载入并执行它。MEX 文件中，MATLAB 是程序的主体，MEX 文件是被 MATLAB 调用的子程序。

MEX 文件主要有以下方面的应用：

- 在 MATLAB 中，M 文件的计算速度特别是循环迭代速度远比 C 程序慢，因此，可以把要求大量循环迭代部分，用 C 语言编写为 MEX 文件，提高计算速度；
- 对已经开发的 C 语言程序，不必转化为 M 文件而重复劳动，通过添加接口程序转为 MEX 文件，由 MATLAB 调用；
- MATLAB 调用 C 语言编写 MEX 文件，可充分发挥 C 语言对低级硬件操作的优越性能，实现对 A/D 采集卡、D/A 输出卡等低级硬件操作。

2. 引擎应用程序

MATLAB 提供了一系列的例程使得别的程序可以调用 MATLAB，从而把 MATLAB 用作一个计算引擎。MATLAB 引擎程序指的是那些通过管道（在 UNIX 系统中）或者 ActiveX（在 Windows 系统中）与独立 MATLAB 进程进行通信的 C/C++或者 Fortran 程序。引擎应用程序中，其他程序如 C/C++程序是主体，MATLAB 程序作为一个计算引擎被调用的对象。

引擎应用程序主要有以下方面的应用：

- 调用 MATLAB 数学函数完成繁重的数学计算，充分发挥 MATLAB 在数值计算上的强大优势，通过 C 语言编写的 GUI 来调用后台的 MATLAB 数学计算功能，完成特殊的需要，将极大缩短程序开发时间；
- 以 MATLAB Compiler 方法将 MATLAB 的数学库或图形库打包成 COM 组件，再与用

户应用程序相连，将使用户程序十分庞大，且 MATLAB Compiler 方法打包不能将 MATLAB 所有函数库打包成 COM 组；而使用引擎应用程序方法，仅需少量的 eng 函数就能全部调用 MATLAB 庞大的数学库和图形库。

3．MAT 文件应用程序

MAT 文件是 MATLAB 专用的用于保存数据至磁盘和向 MATLAB 导入、从 MATLAB 导出数据的数据文件格式。MAT 文件提供了一种简便的机制，它允许在两个平台之间以灵活的方式移动数据。而且，它还提供了一种途径来向其他单机 MATLAB 应用导入或者导出数据。为了简化在 MATLAB 环境之外对 MAT 文件的使用，MATLAB 给出了一个操作例程库，通过它，可以使用 C/C++或者 Fortran 程序读写 MAT 文件。MAT 文件应用程序中，MATLAB 与其他程序具相对独立性，两者间仅通过相对于文本文件而言较为特殊的一种数据文件进行数据交换。

MAT 文件应用程序主要用于 Windows 操作系统与 UNIX 操作系统间传输数据。

7.2 C 语言的 MEX 文件

7.2.1 MEX 文件模式

MEX 文件是用户以其他高级语言（本章内容针对 C 语言而言）编写的，MATLAB 可以像调用 M 文件一样调用的一种动态链接子程序。MATLAB 调用 C 语言的 MEX 文件的流程如图 7.1 所示。

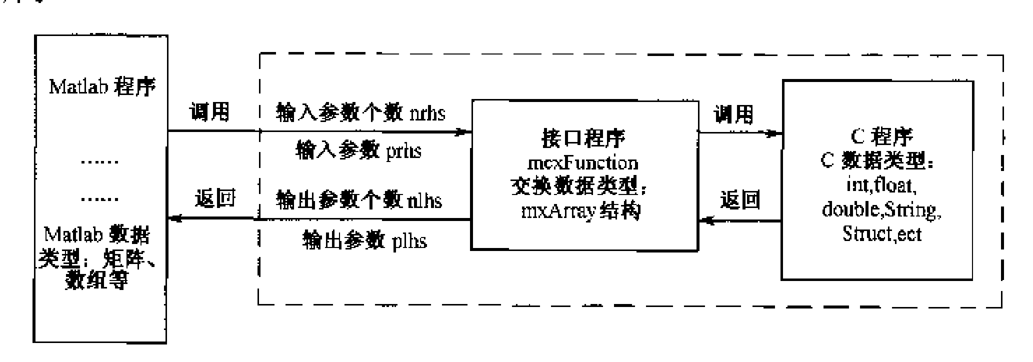


图 7-1　MATLAB 调用 C 语言的 MEX 文件的流程图

由图 7-1 可知，MEX 文件的被调用与执行过程中，其内存管理为 MATLAB 管理机制，调用流程为：

（1）在 MATLAB 命令窗口或 M 文件中调用 MEX 动态链接子程序，操作流程执行 mexFuntion 函数，相关数据由 MATLAB 空间输入至 mexFunction 函数参数 prhs[]；

（2）mexFunction 函数调用 C 程序，操作流程执行 C 程序，相关数据由 mexFunction 函数输出至 C 程序；

（3）执行完 C 程序后，操作流程返回至 mexFunction 函数，相关数据由 C 程序返回至 mexFunction 函数中；

（4）在 mexFunction 函数中，数据传入 plhs[]，mexFunction 执行结束，操作流程返回至 MATLAB，相关数据也由 mexFunction 的输出参数 plhs[]返回至 MATLAB。

MEX 文件编写过程如下：

（1）以 C 语言方式编写用户 C 程序；

（2）编写 MATLAB 与 C 程序间接口函数 mexFunction，并在 mexFunction 中调用 C 程序；

（3）在 MATLAB 命令窗口中设置编译器和工作路径：

```
>>mex –setup
```

MATLAB 提示：

```
Please choose your compiler for building external interface (MEX) files:
Would you like mex to locate installed compilers [y]/n?
```

输入 y 确认后，MATLAB 提示：

```
Select a compiler:
[1] Digital Visual Fortran version 6.0 in C:\Program Files\Microsoft Visual Studio
[2] Lcc C version 2.4 in C:\MATLAB7\sys\lcc
[3] Microsoft Visual C/C++ version 6.0 in C:\Program Files\Microsoft Visual Studio
[0] None
Compiler:
```

输入所需的编译器代号即可，由于 C 程序与 C++程序的兼容性，对 C 语言的 MEX 文件编译，亦可选用 C++编译器，如输入 3，MATLAB 要求确认所选编译器：

```
Please verify your choices:
Compiler: Microsoft Visual C/C++ 6.0
Location: C:\Program Files\Microsoft Visual Studio
Are these correct?([y]/n):y
```

输入 y 确认后，MATLAB 显示本机所设定编译器路径，本机显示为：

```
Try to update options file: C:\Documents and Settings\Administrator\Application Data\MAThWorks
\MATLAB\R14\mexopts.bat
From template: C:\MATLAB7\BIN\WIN32\mexopts\msvc60opts.bat
Done . . .
```

（4）编译 MEX 文件，得到扩展名为.dll 的动态链接子程序；

```
>>mex FileName.c
```

若编译成功，在 MATLAB 工作路径窗口中可见 FileName.dll 动态链接子程序。

（5）在 MATLAB 中像调用 M 文件一样调用所得动态链接子程序。

7.2.2 第一个 MEX 文件

由图 7.1 可知，mexFunction 为 MATLAB 调用 MEX 文件的接口函数，其格式为：

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]);
```

参数 nlhs 和 plhs[]分别为输出参数个数和输出数据，与 MATLAB 一般函数的左手侧参数(left-hand-side) 对应；nrhs 和 prhs[]分别为输入参数个数和输入数据，与 MATLAB 一般函数的右手侧参数(right-hand-side) 对应；

MATLAB 程序中最基本的数据类型是矩阵和数组，而 C 程序中数据类型有整型 int、单精度 float、双精度 double、字符串 string、结构体 struct 及数组等。因此，基于接口函数的 MATLAB 程序与 C 程序间数据交换，最重要的是不同类型数据间的传递。为此，MATLAB 提供了一个特殊的 C 语言结构——mxArray，以在 C 语言中表示 MATLAB 的矩阵和数组。在 MEX 文件的接口程序中，输出参数 plhs[]和输入参数 prhs[]均为 mxArray 定义的对象。

MATLAB 在调用 MEX 文件时，将数据传给 mexFunction 的输入参数 prhs[]，mexFunction 在返回 MATLAB 之前，将 C 程序的计算结果传入输出参数 plhs[]中，实现 MATLAB 与 C 程序数据交换。此外，由于 MATLAB 与 C 语言的内存管理机制不同，MEX 文件编写过程中，应统一应用 MATLAB 内存管理机制，即在接口程序与用户 C 程序中，内存的分配与管理，均采用 mx 内存管理函数，如 C 程序中 malloc、realloc、calloc 和 free 函数，一般以 mxMalloc、mxRealloc、mxCalloc 和 mxFree 函数代替。

综上，我们编写了第一个 MEX 文件“h1.c”来开始学习如何使用 MATLAB 提供的 API。

```
//////////////////MEX 头文件////////////////////////////////////////
#include "mex.h"
/////////////////用户 C 程序 /////////////////////////////////////////
double OutPut(int n,double *rP)
{ int i;   double *sum,sum0;
FILE *fp1;
sum=mxCalloc(1,sizeof(double));     ////////////////内存的管理机制为 MATLAB 机制
if((fp1=fopen("OutPut1.txt","w"))==NULL){ mexErrMsgTxt(" Can't open OutPutfile\n");}
*sum=0.0;
fprintf(fp1,"Input  %d Data is:",n);
for(i=0;i<n;i++){*sum=*sum+rP[i];fprintf(fp1,"  %7.2lf ",rP[i]);}
fprintf(fp1,"\nsum=  %7.2lf \n",*sum);
fclose(fp1);
sum0=*sum;
mxFree(sum);//////////内存的管理机制为 MATLAB 机制
return sum0; ////// C 程序返回计算值
}
///////////////  接口程序 mexFunction//////////////////////////////////
void mexFunction(int nlhs,mxArray *plhs[],int nrhs, mxArray *prhs[])
{int i; double *rP,*lP,sum;
///////////////  由   prhs[]   中取得 MATLAB 输入数据
rP=mxCalloc(nrhs,sizeof(double));    //////////内存的管理机制为 MATLAB 机制
for(i=0;i<nrhs;i++)rP[i]=mxGetScalar(prhs[i]);
//////////////   调用 C 程序
sum=OutPut(nrhs,rP);
/////////////    进行数据类型转换，并存入 plhs[] 返回 MATLAB
nlhs=nrhs;
plhs[0]=mxCreateDoubleMATrix(1,nlhs,mxREAL);
lP=mxGetPr(plhs[0]);
for(i=0;i<nlhs;i++)lP[i]=i*sum;
mxFree(rP);    //////////内存的管理机制为 MATLAB 机制
mxFree(lP);
}
////////////////结束////////////////////////////////////////////////
```

该程序包括以下几部分：

（1）MEX 程序的头文件“mex.h”；

（2）用户 C 程序 OutPut()；

（3）接口程序 mexFunction()。

MATLAB 在调用该 MEX 文件时，首先在 H1.c 中寻找接口函数 mexFunction()，并将数据

传至 mexFunction 的输入参数 prhs[]。由于 prhs[]是由 mxArray 定义的指针型数组，MATLAB 将输入参数地址赋给 prhs[]，将输入参数个数赋给 nrhs，用户无须对 prhs[]赋起始地址。并且，MATLAB 提供了一系列函数来获取输入参数 prhs[]的地址和数据。如本例中，mxGetScalar()函数来获取 prhs[]中的标量，并赋给 C 语言双精度实型一维数组 rP。

对于输出参数 plhs[]，接口函数 mexFunction 参数列表中仅对其数据类型进行了定义，但 mexFuntion 未给参数 plhs[]赋地址，用户在使用该参数前须赋起始地址。如本例中，以 mxCreateDoubleMATrix 将 plhs[0]赋为 mxArray 双精度实型矩阵指针，所指向矩阵行数为 1，列数由输入参数 nrhs 个数确定。由于 plhs[]为 mxArray 定义的指针数组，MATLAB 也提供了一系列函数以便用户将数据赋给 plhs[]。如本例中 mxGetPr()函数将 plhs[0]起始地址赋给 C 语言的双精度实型一维数组 lP，只要用户将所得数据赋给 lP，mexFunction 函数通过 plhs[0]将数据从 MEX 文件中输出至 MATLAB，实现数据交换。

在 MATLAB 命令窗口中将“h1.c”文件编译后，得动态链接子程序 h1.dll，并在 MATLAB 命令行中调用。

```
>>mex h1.c
>>a=h1(1,2,3,4,5)
a=
      0     15     30     45     60
```

同时，由于用户 C 程序中执行了文件输出，在当前目录下生成“OutPut1.txt”文件，输出结果为：

```
Input 5 Data is:    1.00    2.00    3.00    4.00    5.00
sum=    15.00
以不同参数调用 h1.dll，如
>> b=hh1(2,4)
b =
      0      6
```

当前目录下生成“OutPut1.txt”文件，输出结果为：

```
Input 2 Data is:    2.00    4.00
sum=    6.00
```

7.2.3 不同数据类型的传递

1．矩阵

矩阵是 MATLAB 最基本的数据类型之一，但 MATLAB 中矩阵与 C 语言中二维数组有较大差别。首先，MATLAB 中矩阵是按列方式存储的，而 C 语言的二维数组则是按行方式存储的；其次，MATLAB 中矩阵下标从 1 开始，而 C 语言的二维数组下标从 0 开始。因此，MATLAB 矩阵常转换为 C 语言一维数组，而较少采用二维数组。MATLAB 矩阵与 C 语言一维数组间的转换公式如下。

C_array[j*M+i]=M_matrix(i,j)

式中，M 为 MATLAB 矩阵 M_matrix 的行数。

将“h1.c”作一定改进后，得“h2.c”MEX 文件，该文件以 MATLAB 最常用数据结构（矩阵）为输入参数。在调用 MEX 文件时，MATLAB 将矩阵型参数赋给 prhs[]，用户通过 rP=mxGetPr(prhs[i])函数获取指向矩阵 prhs[i]的指针，并赋给 C 一维数组 rP，然后再通过

M=mxGetM(prhs[i])和 N=mxGetN(prhs[i])获取 prhs[i]矩阵的行数 M 与列数 N，这样就实现 MATLAB 矩阵至 C 一维数组的转换；C 程序执行完成后，用户先对 plhs[i]赋地址，再将 plhs[i]地址赋给 C 一维数组 lP，这样，用户只要对 C 一维数组 lP 操作，相应操作结果就存入了 plhs[i]中，实现 C 一维数据至 MATLAB 矩阵数据转换。程序清单如下。

```
//////////////////MEX 头文件////////////////////////////////////////
#include "mex.h"
#include "math.h"
//////////////////用户 C 程序 //////////////////////////////////////
void OutPut(FILE* fp1,int M,int N,double *rP,int ii)
{int i,j;
fprintf(fp1,"\n The  %d Input    Data is:\n",ii);
for(i=0;i<M;i++)
{
     for(j=0;j<N;j++)fprintf(fp1,"    %7.2lf",rP[j*M+i]);
     fprintf(fp1,"\n");
}
}
////////////////  接口程序 mexFunction//////////////////////////////
void mexFunction(int nlhs,mxArray *plhs[],int nrhs, mxArray *prhs[])
{ int i,M,N,ii,jj;
   double *rP,*lP;
FILE *fp1;
if((fp1=fopen("OutPut2.txt","w"))==NULL){ mexErrMsgTxt(" Can't open OutPutfile\n");}
nlhs=nrhs;
for(i=0;i<nrhs;i++){
rP=mxGetPr(prhs[i]);        /////获取指向矩阵 prhs[i]的指针
M=mxGetM(prhs[i]);
N=mxGetN(prhs[i]);        ///获取 prhs[i]矩阵的行数 M 与列数 N
OutPut(fp1,M,N,rP,i);     ////////////////  调用 C 程序
plhs[i]=mxCreateDoubleMATrix(M,N,mxREAL);      //地址赋初值
lP=mxGetPr(plhs[i]);      /////获取指向矩阵 plhs[i]的指针
for(ii=0;ii<M;ii++)for(jj=0;jj<N;jj++)lP[jj*M+ii]=sqrt(rP[jj*M+ii]);      //通过 lP 向 plhs[i]赋值
}
fclose(fp1);
}
//////////////////结束//////////////////////////////////////////////
```

在 MATLAB 命令窗口中将“h2.c”文件编译后，得动态链接子程序 h2.dll，并在 MATLAB 命令行中调用。

```
>> mex h2.c
>> a=1:8
>> b=[a;3*a;5*a]
>> [L1,L2,L3]=h2(b,2*b,3*b)
L1 =
    1.0000    1.4142    1.7321    2.0000    2.2361    2.4495    2.6458    2.8284
    1.7321    2.4495    3.0000    3.4641    3.8730    4.2426    4.5826    4.8990
    2.2361    3.1623    3.8730    4.4721    5.0000    5.4772    5.9161    6.3246
L2 =
    1.4142    2.0000    2.4495    2.8284    3.1623    3.4641    3.7417    4.0000
```

```
    2.4495    3.4641    4.2426    4.8990    5.4772    6.0000    6.4807    6.9282
    3.1623    4.4721    5.4772    6.3246    7.0711    7.7460    8.3666    8.9443
L3 =
    1.7321    2.4495    3.0000    3.4641    3.8730    4.2426    4.5826    4.8990
    3.0000    4.2426    5.1962    6.0000    6.7082    7.3485    7.9373    8.4853
    3.8730    5.4772    6.7082    7.7460    8.6603    9.4868   10.2470   10.9545
```

同时，由于用户 C 程序中执行了文件输出，在当前目录下生成“OutPut2.txt”文件，输出结果为：

```
The 0 Input   Data is:
    1.00    2.00    3.00    4.00    5.00    6.00    7.00    8.00
    3.00    6.00    9.00   12.00   15.00   18.00   21.00   24.00
    5.00   10.00   15.00   20.00   25.00   30.00   35.00   40.00
 The 1 Input   Data is:
    2.00    4.00    6.00    8.00   10.00   12.00   14.00   16.00
    6.00   12.00   18.00   24.00   30.00   36.00   42.00   48.00
   10.00   20.00   30.00   40.00   50.00   60.00   70.00   80.00
 The 2 Input   Data is:
    3.00    6.00    9.00   12.00   15.00   18.00   21.00   24.00
    9.00   18.00   27.00   36.00   45.00   54.00   63.00   72.00
   15.00   30.00   45.00   60.00   75.00   90.00  105.00  120.00
```

2. 字符串

MATLAB 和 C 程序间同样可以传递字符串类型的变量。对“h2.c”文件作一定改动后，得“h3.c”MEX 文件。该文件中，MATLAB 传入字符串变量至 prhs[]中，mexFunction 将字符串变量传入 C 字符串 rP_buf 中，后调用 C 程序进行字符串逆操作和操作结果输出，同时将操作结果返回 mexFunction 输出参数 plhs[]，以返回 MATLAB。程序清单如下。

```
//////////////////MEX 头文件//////////////////////////////////////////
#include "mex.h"
#include "String.h"
//////////////////用户 C 程序 /////////////////////////////////////////
void reword(FILE* fp1,char *rP_buf,char *lP_buf,int buf,int ii)
{int i;
fprintf(fp1,"\n The  %d Input    String is:  %s\n",ii,rP_buf);
for(i=0;i<buf-1;i++){*(lP_buf+i)=*(rP_buf+buf-i-2);}
fprintf(fp1,"\n The  %d Output    String is:  %s\n",ii,lP_buf);
}
///////////////   接口程序 mexFunction////////////////////////////////
void mexFunction(int nlhs,mxArray *plhs[],int nrhs, mxArray *prhs[])
{int i,j,buf;
 char *rP_buf,*lP_buf;
FILE *fp1;
if((fp1=fopen("OutPut3.txt","w"))==NULL){ mexErrMsgTxt(" Can't open OutPutfile\n");}
nlhs=nrhs;
for(i=0;i<nrhs;i++){
if(mxIsChar(prhs[i])!=1)mexErrMsgTxt("Input must be a string");//参数检查
buf=(mxGetM(prhs[i])*mxGetN(prhs[i]))+1; //////获取每一个输入参数对应的字符串大小，
                                          ///由于 CString 以 NULL 作为字符串结束标志，故加 1
rP_buf=mxCalloc(buf,sizeof(char));        //////MATLAB 内存管理机制
```

```
lP_buf=mxCalloc(buf,sizeof(char));        /////MATLAB 内存管理机制
if(mxGetString(prhs[i],rP_buf,buf)==1)   //从 prhs[i]中获取输入参数
mexWarnMsgTxt("Not enough space.String is truncated");
reword(fp1,rP_buf,lP_buf,buf,i);         ////调用 C 程序
plhs[i]=mxCreateString(lP_buf);          //plhs[i]地址赋初值
mxFree(rP_buf);   ///MATLAB 内存管理机制
mxFree(lP_buf);
}
fclose(fp1);
}
///////////////结束//////////////////////////////////////////////
```

在 MATLAB 命令窗口中将“h3.c”文件编译后，得动态链接子程序 h3.dll，并在 MATLAB 命令行中调用。

```
>> mex h3.c
a='my book is yellow';
b='school is over';
c='the sky is blue';
[L1,L2,L3]=h3(a,b,c)
L1 =
wolley si koob ym
L2 =
revo si loohcs
L3 =
eulb si yks eht
```

同时，由于用户 C 程序中执行了文件输出，在当前目录下生成“OutPut3.txt”文件，输出结果为：

```
The 0 Input    String is: my book is yellow
The 0 Output   String is: wolley si koob ym
The 1 Input    String is: school is over
The 1 Output   String is: revo si loohcs
The 2 Input    String is: the sky is blue
The 2 Output   String is: eulb si yks eht
```

3. 结构体与数组

与 C 语言一样，MEX 文件结构体的数据是由若干记录组成，每一个记录保存在结构体的字段中，结构保存的数据可以是任意类型的 MATLAB 数据。在 MEX 文件中，创建结构体的基本过程如下。

（1）准备不同字段的数据：可以由 MATLAB 创建，也可由 C 语言创建。

（2）创建结构体数据对象指针：

```
mxArray *Data=mxCreateStructMATrix(int m, int n, int nFields, const char **FieldsName);
```

参数：m mxArray　类型矩阵的行数；

n mxArray　类型矩阵的列数；

nFileds　每一结构体的字段数；

FieldsName　每一结构体的字段名。

（3）使用 mx 函数完成结构体数据赋值：

mxSetField(mxArray *Data, int Num, const char **FieldsName, (type) Value);

参数：Data　所创建的 mxArray 类型结构体；

Num　整个结构体的序号；

FieldsName　结构体中的字段名；

Value　与结构体中对应字段中数据类型相一致的数据。

下面以“h4.c”来说明 MEX 文件中如何创建结构体及其数据返回 MATLAB 的实现过程。

```
//////////////////MEX 头文件//////////////////////////////////////////
#include "mex.h"
//////////////////用户 C 程序:C 语言定义结构体 ////////////////////////////
struct Person
{char Name[10];
 int Age;
 char Sex[2];
 char Address[30];
 char Tel[15];
};
//////////用户 C 程序:C 语言输入结构体数据 //////////////////////////////////
void InputPerson(int n,struct Person *P)
{ int i;char *InputFile;
FILE *fp1;
mexPrintf("Input The FileName Is Per.txt \n");
InputFile="Per.txt";
if((fp1=fopen(InputFile,"r"))==NULL){ mexErrMsgTxt(" Can't open InPutfile\n");}
for(i=0;i<n;i++)fscanf(fp1,"%s %d %s %s %s",
                        P[i].Name,&P[i].Age,P[i].Sex,P[i].Address,P[i].Tel);
fclose(fp1);
}

////////////////  接口程序 mexFunction/////////////////////////////////
void mexFunction(int nlhs,mxArray *plhs[],int nrhs, mxArray *prhs[])
{int i,n;
struct Person *P;
mxArray *Data;
char **Fields;////mx 结构体字段名
/////////////// 由  prhs[]  中取得 MATLAB 输入数据
n=(int)mxGetScalar(prhs[0]);
P=mxCalloc(n,sizeof(struct Person));///mx 内存分配方式为结构体分配内存
Fields[0]="Name"; Fields[1]="Age";
Fields[2]="Sex";  Fields[3]="Address";
Fields[4]="Tel";

InputPerson(n,P); ////调用 C 程序
Data=mxCreateStructMATrix(1,1,5,Fields);///创建 mx 结构体
for(i=0;i<n;i++){  /////为 mx 结构体赋值
mxSetField(Data,0,Fields[0],mxCreateString(P[i].Name));
mxSetField(Data,0,Fields[1],mxCreateDoubleScalar((double)(P[i].Age)));
mxSetField(Data,0,Fields[2],mxCreateString(P[i].Sex));
mxSetField(Data,0,Fields[3],mxCreateString(P[i].Address));
```

```
mxSetField(Data,0,Fields[4],mxCreateString(P[i].Tel));
plhs[i]=mxCreateStructMATrix(1,1,5,Fields);///将输出参数创建为 mx 结构体并分配内存
plhs[i]=mxDuplicateArray(Data);////mxArray 对象一结构体数值复制
}
mxDestroyArray(Data); ///释放 mxArray 对象内存
mxFree(P);///////mx 内存释放方式将结构体内存释放
}
/////////////////结束////////////////////////////////////////////////
```

“h4.c”程序中，首先由 C 语言定义了结构体 Person，共包括 5 个字段：Name、Age、Sex、Address 和 Tel，各字段数据事先保存在当前目录的“Per.txt”文件中。

在 MATLAB 命令窗口中将“h4.c”文件编译后，得动态链接子程序 h4.dll，并在 MATLAB 命令行中调用。

```
>> mex h3.c
>>[L1,L2,L3]=h4(3)
Input The FileName Is Per.txt
L1 =
        Name: 'Liuh'
         Age: 29
         Sex: 'M'
     Address: 'Street3'
         Tel: '1390000'
L2 =
        Name: 'Li'
         Age: 29
         Sex: 'M'
     Address: 'Street4'
         Tel: '1360000'
L3 =
        Name: 'Su'
         Age: 30
         Sex: 'M'
     Address: 'Street5'
         Tel: '1350000'
```

从“h4.c”中可知，在 MATLAB 中调用 h4.dll 动态链接子程序后，操作流程为：

（1）操作流程寻找接口函数 mexFunction 后，从输入参数 prhs[0]中取得需读入结构体记录数为 3，并为结构体分配相应的内存空间；

（2）为结构赋于字段名；

（3）调用用户 C 程序，并从中读取结构体各字段的数据；

（4）流程返回接口程序；

（5）在接口程序中创建 mxArray 类型结构体；

（6）为 mxArray 类型结构体各字段赋由 C 程序中取得的数据；

（7）以矩阵数据复制方式将结构体数据复制至接口程序输出参数 plhs[i]中；

（8）释放内存，返回 MATLAB。

7.2.4 MEX 文件内存管理

MATLAB 与 C 语言内存管理机制存在显著差别。MATLAB 采用自动内存释放机制，即在 M 文件或 MEX 文件中，内存的分配与释放，主要由 MATLAB 的解释器来自动完成。但在 C 语言程序中，内存的分配与释放由程序员编写代码来完成。因此，在 MATLAB 调用 C 程序的 MEX 文件中，内存管理仍显得十分重要。下面分别介绍 MEX 文件中内存分配与释放函数。

（1）mxMalloc

void *mxMalloc(size_t n);

参数：n　需分配的字节数。

返回：若分配成功，则返回一个指向所分配动态内存起始地址的指针；否则，终止 MEX 流程并返回 MATLAB。

说明：mxMalloc 函数自动分配 n 字节的连续内存空间，并在 MATLAB 内存管理器中注册返回堆栈入口（returned heap space）。MATLAB 内存管理器有 MEX 文件中，所有由 mxMalloc 函数分配的内存注册清单，当 MEX 文件执行完毕或意外中止时，流程返回 MATLAB 前，MATLAB 内存管理器将自动释放该清单上的所有内存空间。

内存释放函数：void mxFree(void *ptr);

（2）mxCalloc

void *mxCalloc(size_t n, size_t size);

参数：n　需分配的元素个数；

size　所分配每一元素字节数，通常由 sizeof()函数取得。

返回：若分配成功，则返回一个指向所分配动态内存起始地址的指针；否则，终止 MEX 流程并返回 MATLAB。

说明：mxCalloc 函数自动分配 n*size 字节的连续内存空间，并将这 n 个元素初值均赋为 0，同时也在 MATLAB 内存管理器中注册返回堆栈入口（returned heap space）。MATLAB 内存管理器有 MEX 文件中所有由 mxCalloc 函数分配的内存注册清单，当 MEX 文件执行完毕或意外中止时，流程返回 MATLAB 前，MATLAB 内存管理器将自动释放该清单上的所有内存空间。

内存释放函数：void mxFree(void *ptr);

（3）mxRealloc

void *mxRealloc(void *ptr, size_t size);

参数：ptr　需重新分配内存的指针；

size　需重新分配内存的字节数。

返回：若分配成功，则返回一个指向所分配动态内存起始地址的指针；否则，返回 0。

说明：mxRealloc 函数对 ptr 重新分配内存，即使操作失败，也应进行内存释放。否则，易产生内存泄漏。

内存释放函数：void mxFree(void *ptr);

（4）mxCreate*函数

MEX 文件中，MATLAB 提供了众多的 mxCreate 函数来创建不同类型的 mxArray 放据类型对象。这些函数操作成功后，返回相应的内存地址（指针）；否则，返回不同类型的错误信息。由 mxCreate*函数创建的 mxArray 数据类型所占内存空间，均由 mxDestroyArray()函数释放。

常用的 mxCreate 函数主要有如下几种。

① mxArray * mxCreateDoubleScalar (double value);

参数：value　所创建的 mxArray 数据类型（1 行×1 列）的初值。

返回：若操作成功，则返回一个指向所创建的 mxArray 数据类型（1 行×1 列）的指针；否则，终止 MEX 流程并返回 MATLAB。

说明：该函数与下述语句作用相同，均可返回一个指向 1 行×1 列的 mxArray 类型指针。

```
a = mxCreateDoubleMATrix(1, 1, mxREAL);
*mxGetPr(pa) = value;
```

内存释放函数：void mxDestroyArray(mxArray *array_ptr);

② mxArray *mxCreateString(const char *str);

参数：str　所创建的 mxArray 类型的字符串（仅 1 行）的初值。

返回：若操作成功，则返回一个指向所创建的 mxArray 类型的字符串指针；否则，返回 NULL。

内存释放函数：void mxDestroyArray(mxArray *array_ptr);

③ mxArray *mxCreateNumericMATrix(int m, int n, mxClassID class, mxComplexity ComplexFlag);

参数：m　需创建的 mxArray 矩阵的行数；

n　需创建的 mxArray 矩阵的列数；

class　数据类型，取值为：mxDOUBLE_CLASS, mxSINGLE_CLASS, mxINT8_CLASS, mxUINT8_CLASS, mxINT16_CLASS, mxUINT16_CLASS, mxINT32_CLASS, mxUINT32_CLASS, mxINT64_CLASS, and mxUINT64_CLASS，分别对应双精度、单精度、8 位整型、8 位无符号整型、16 位整型、16 位无符号整型、32 位整型、32 位无符号整型、64 位整型、64 位无符号整型等数据类型；

ComplexFlag　取值为 mxREAL 和 mxCOMPLEX，分别代表所创建的 mxArray 矩阵为实数型矩阵和复数型矩阵。

返回：若操作成功，则返回一个指向所创建的 mxArray 类型矩阵的指针；否则，终止 MEX 流程并返回 MATLAB。

内存释放函数：void mxDestroyArray(mxArray *array_ptr);

④ mxArray *mxCreateDoubleMATrix(int m, int n, mxComplexity ComplexFlag);

参数：m　需创建的 mxArray 矩阵的行数；

n　需创建的 mxArray 矩阵的列数；

ComplexFlag　取值为 mxREAL 和 mxCOMPLEX，分别代表所创建的 mxArray 矩阵为实数型矩阵和复数型矩阵。

返回：若操作成功，则返回一个指向所创建的 mxArray 类型矩阵的指针；否则，终止 MEX 流程并返回 MATLAB。

内存释放函数：void mxDestroyArray(mxArray *array_ptr);

⑤ mxArray *mxCreateCellMATrix(int m, int n);

参数：m　需创建的 mxArray 矩阵的行数；

n　需创建的 mxArray 矩阵的列数。

返回：若操作成功，则返回一个指向所创建的 cell mxArray 类型的指针。否则，终止 MEX

流程并返回 MATLAB。

内存释放函数：void mxDestroyArray(mxArray *array_ptr);

MEX 文件内存分配与释放例子，可参见“h1.c”、“h2.c”和“h3.c”。在给 MEX 文件的输出参数 plhs[]赋值，除各例中所采取赋值方式外，还可使用 memcpy()函数或 mxDuplicateArray()函数，进行数据复制。

“h1.c”文件中，在使用 plhs[0]=mxCreateDoubleMATrix(1,nlhs,mxREAL)给 plhs[0]分配内存地址后，对应的内存数据复制 memcpy()函数为：

```
lP= mxCalloc(nlhs,sizeof(double));
for(i=0;i<nlhs;i++)lP[i]=i*sum;
memcpy(mxGetPr(plhs[0]),lP,nlhs*sizeof(double));    ///使用内存数据复制函数 memcpy()
mxFree(lP);
```

“h1.c”文件中，在使用 plhs[0]=mxCreateDoubleMATrix(1,nlhs,mxREAL)给 plhs[0]分配内存地址后，对应的 mxDuplicateArray()函数为：

```
mxArray lP0;  ///定义 mxArray 类型的指针变量*lP0
lP0=mxCreateDoubleMATrix(1,nlhs,mxREAL);    /////并对 lP0 分配内存地址
lP= mxCalloc(nlhs,sizeof(double));
for(i=0;i<nlhs;i++)lP[i]=i*sum;
memcpy(mxGetPr(lP0),lP,nlhs*sizeof(double));   /////采用内存复制函数将数据复制与 lP0
plhs[0]=mxDuplicateArray(lP0);    /////使用内存数据复制函数 mxDuplicateArray()
mxFree(lP);
mxDestroyArray(lP0);    ///////释放 lP0
```

在 MEX 文件内存管理中，MATLAB 还提供了内存变量保留机制，用户可以声明 MEX 文件中的变量或某一段内存为保留属性，方法是使用 mexMakeArrayPersistent 或 mexMakeMemeoryPersistent。由于 MATLAB 内存自动释放机制不能释放用户申明的具保留属性的变量和内存段，在 MEX 整个文件过程甚至 MEX 文件终止后，这些被保留的变量和内存段一直不会被自动释放，用户可在整个 MEX 文件中使用这些变量和内存，但当 MEX 文件终止时，用户必须自己释放这些变量和内存段。为防止因 MEX 文件意外终止而导致操作进程不能执行用户编写的内存释放代码，MATLAB 提供了 mexAtEixt 函数，该函数在 MEX 文件在内存中被清除时调用，因此，用户对申明了保留属性的变量和内存段的释放代码，一般通过该函数调用执行，以避免内存泄漏。

```
///////////////////////////////////   mexAtEixt 执行时调用的函数
void DeAllocPersistent()
{
mexPrintf("\n 退出 MEX 文件，清除内存!\n");
mxDestroyArray(lP0);
mxFree(buf);
}
////////////////   接口程序 mexFunction/////////////////////////////////
void mexFunction(int nlhs,mxArray *plhs[],int nrhs, mxArray *prhs[])
{
mxArray *lP0;
char *buf;
...............
lP0=mxCreateDoubleMATrix(1,nlhs,mxREAL);
mexMakeArrayPersistent(lP0); ////// 申明 lP0 为保留属性
```

```
……………………
buf= mxMalloc(100);
mexMakeMemoryPersistent(buf); ////////// 申明 buf 为保留属性
……………………
mexAtEixt(DeAllocPersistent); //////////// MEX 文件在内存中被清除时调用 mexAtExit 函数，
///// 一般一个 MEX 文件中只一个 mexAtExit 函数，若用户提供多个
///// mexAtExit 函数时，MATLAB 只调用最近的 mexAtExit 函数
……………………
}
```

最后值得一提的 mxDestroyArray(mxArray *array_ptr)函数，该函数不仅释放了 array_ptr 所指的 mxArray 数据的 m 行和 n 列空间，还释放了所有与之相关的诸如实部指针 pr、虚部指针 pi 等内存地址。因此，对于 MEX 文件中的输出参数，一般不使用 mxDestroyArray()函数来释放其内存，而由 MATLAB 的自动内存管理机制来释放。

7.2.5 MEX 文件调试

与其他语言编程一样，用户编写的 MEX 文件很难保证一次成功，往往需要进行大量的调试工作。MATLAB 提供了不同操作系统下调试 MEX 文件的方法。鉴于 Visual C++ 的强大调试功能及其对 C 程序的兼容性，本节以“H1.c”为例，简要介绍 Microsoft Visual Studio 6 下 MEX 文件的调试方法。

“h1.c”文件的调试步骤如下。

（1）在 MATLAB 命令窗口中，以-g 选项编译所调试的文件，MATLAB 自动为所生成的 MEX 文件添加必要的调试信息。

```
>>mex –g h1.c
```

（2）在 Windows 命令行（开始→运行）输入“msdev D:\MATLAB_C\mex\h1.dll”，Windows 启动 Microsoft Visual Studio 6 并打开了 h1.c 文件，其中“D:\MATLAB_C\mex\h1.dll”为本机“h1.c”文件存储目录。

（3）打开 Project→Settings…设置框，在“Settings For Debug”对话框中，将“Executable for debug session”设为“C:\MATLAB7\bin\win32\MATLAB.exe”，其中“C:\MATLAB7”为本机 MATLAB 安装目录（图 7-2）。

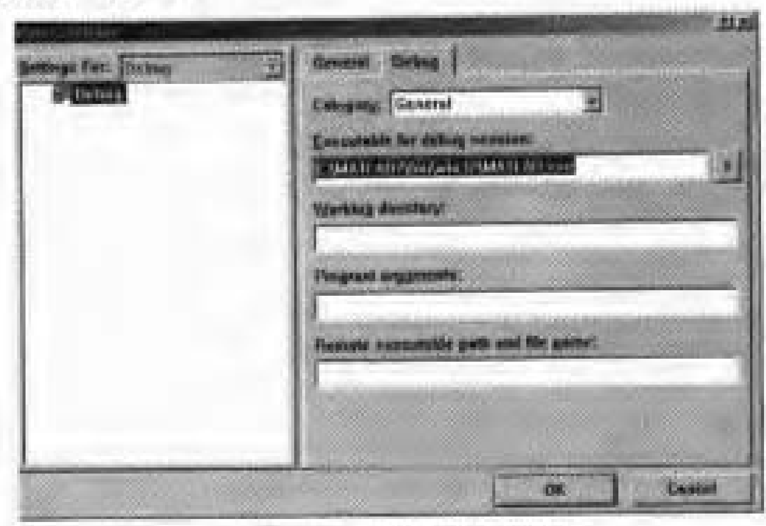


图 7-2　MEX 文件调试属性设置

（4）在 Microsoft Visual Studio 6 打开的“h1.c”中，将光标移至需检查的行上，设置断点、执行调试（图 7-3）。当用户按下调试按钮后，Microsoft Visual Studio 6 启动 MATLAB，用户需在 MATLAB 命令窗口中调用“h1.dll”，MEX 执行至用户所设断点时，流程停止于 Microsoft Visual Studio 6 的“h1.c”断点处，用户再次按下调试按钮后，流程继续执行并停止于下一断点，如此循环，直至所有断点完毕。

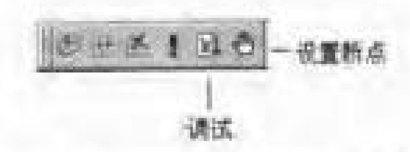
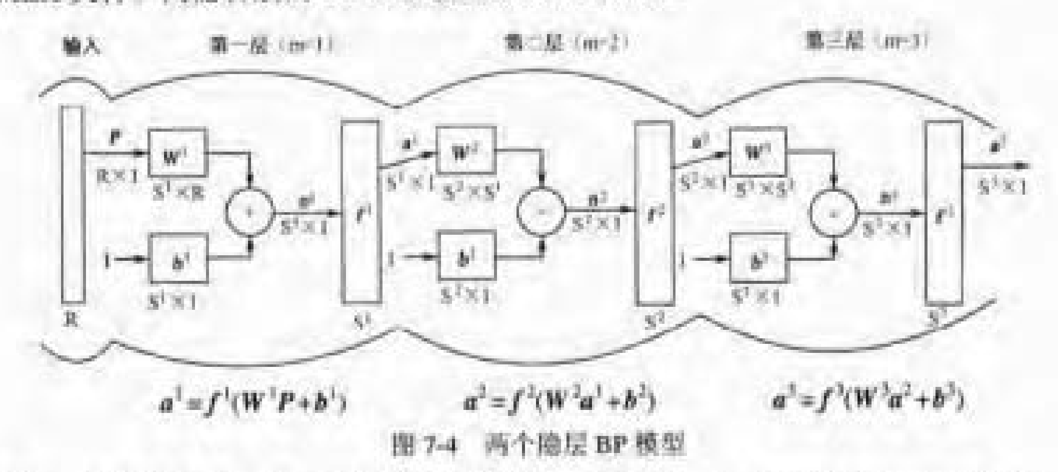


图 7-3　Microsoft Visual Studio 6 调试菜单

7.2.6　MEX 应用程序开发实例

本节采用两隐含层的人工神经网络预测 C 语言程序为例，说明如何在原有 C 程序基础上编写 MEX 文件。两隐含层的 BP 网络模型如图 7-4 所示。



$a^1=f^1(W^1P+b^1)$　　$a^2=f^2(W^2a^1+b^2)$　　$a^3=f^3(W^3a^2+b^3)$

图 7-4　两个隐层 BP 模型

其中：f^i 传递函数，第一层和第二层取为'tansig'函数，第三层取为'purelin'函数；

W^i、b^i 分别为第 i 层权值和阈值；

S^1、S^2、S^3 分别为第 1、2、3 层神经元数目；

P 输入向量；

a^1、a^2、a^3 分别为第 1、2、3 层输出结果。

训练过程中的数学模型：

$$\Delta W^m(k)=\gamma\Delta W^m(k-1)-(1-\gamma)\alpha A^m(a^{m-1})^T$$
$$\Delta b^m(k)=\gamma\Delta b^m(k-1)-(1-\gamma)\alpha A^m$$
$$W^m(k+1)=W^m(k)+\Delta W^m(k)$$
$$b^m(k+1)=b^m(k)+\Delta b^m(k)$$

式中：γ 是动量系数，α 是学习率，满足

$$0\leqslant\gamma<1 \qquad 0\leqslant\alpha<1$$

A^m 为 m 层的敏感性

“Bp3.c”程序清单如下。

```
////////////////头文件
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "mex.h"
```

```
#define DN 30        // Number of Neural net cells
#define DM 500       // Number of Samples
#define DL 20        // Output frequency
#define LP 0.125   // Learning velocity
#define NSS 1000     //

///////////////////用户 C 程序-start ///////////////////////////
///////////////////////////// f1 is logsig function;   f3 is pureline function;
double f1(double x){return (double)(1/(1+exp(-x)));}
double df1(double x){double tp; tp=(double)exp(-x);return tp/((1+tp)*(1+tp));}
double f3(double x){return x;}
double df3(double x){return 1.0;}

/////////////////////////////////////////////////////
Bp3_C()
{
int i,j,s0,s1,s2,s3,k,n,jj,n0,NStep;
double E,arfa,gama,err,err0;
double pp[DN][DM],tt[DN][DM],tt0[DN],ee[DN];
double aa0[DN],aa1[DN],aa2[DN],aa3[DN],ss1[DN],ss2[DN],ss3[DN],nn1[DN],nn2[DN],nn3[DN];
double bb1[DN],bb2[DN],bb3[DN],dbb10[DN],dbb20[DN],dbb30[DN],dbb1[DN],dbb2[DN];
double dbb3[DN];
double ww1[DN][DN],ww2[DN][DN],ww3[DN][DN];
double dww1[DN][DN],dww2[DN][DN],dww3[DN][DN];
double dww10[DN][DN],dww20[DN][DN],dww30[DN][DN];
//////////////
double kk[NSS],Err[NSS]; int k_DL;
mxArray *rP[1],*lP[1],*rrP[1],*llP[1];
///////////////////////////////////////////////////////

char *DataFile,*Fout,*Fss;FILE *fp0,*fp1,*fp2;
err0=5.0; E=(double)1e-10;   arfa=LP; gama=0.45;
/////////////////////////////////////////////

DataFile="J6.txt"; ///////////////////////////////输入文件名
//scanf("%s",DataFile);
if((fp0=fopen(DataFile,"r"))==NULL){mexErrMsgTxt(" Can't open OutPutfile\n");}
Fout=strdup(DataFile);Fss="-r.txt";strcat(Fout,Fss);
if((fp1=fopen(Fout,"w"))==NULL){mexErrMsgTxt(" Can't open OutPutfile\n");}
Fout=strdup(DataFile);Fss="-rr.txt";strcat(Fout,Fss);
if((fp2=fopen(Fout,"w"))==NULL){mexErrMsgTxt(" Can't open OutPutfile\n");}

fscanf(fp0,"%d%d%d%d%d%d%lf%lf%d",&n,&n0,&s0,&s1,&s2,&s3,&arfa,&gama,&NStep);
for(jj=0;jj<n;jj++){
for(i=0;i<s0;i++){fscanf(fp0,"%lf",&pp[i][jj]);}   // pp[][] is Input   Sample
for(i=0;i<s3;i++){fscanf(fp0,"%lf",&tt[i][jj]);}   // tt[][] is target Sample
}     n=n0; if(NStep>=NSS*DL)NStep=NSS*DL-1;

///////////////  Initial WW & BB   within(0,1)
for(i=0;i<s1;i++){bb1[i]=rand()/32767.0;for(j=0;j<s0;j++)ww1[i][j]=rand()/32768.0;}
```

```
for(i=0;i<s2;i++){bb2[i]=rand()/32767.0;for(j=0;j<s1;j++)ww2[i][j]=rand()/32768.0;}
for(i=0;i<s3;i++){bb3[i]=rand()/32767.0;for(j=0;j<s2;j++)ww3[i][j]=rand()/32768.0;}
for(i=0;i<s1;i++){dbb10[i]=0.0;for(j=0;j<s0;j++)dww10[i][j]=0.0;}
for(i=0;i<s2;i++){dbb20[i]=0.0;for(j=0;j<s1;j++)dww20[i][j]=0.0;}
for(i=0;i<s3;i++){dbb30[i]=0.0;for(j=0;j<s2;j++)dww30[i][j]=0.0;}
//////////////////////////////////////////////////
k_DL=k=0;
while(1){
for(jj=0;jj<n;jj++){
for(i=0;i<s0;i++)aa0[i]=pp[i][jj];
for(i=0;i<s3;i++)tt0[i]=tt[i][jj];
////////////////   Prepropagating aa0[] to aa3[]
for(i=0;i<s1;i++){nn1[i]=0.0;for(j=0;j<s0;j++)nn1[i]=nn1[i]+ww1[i][j]*aa0[j];}
for(i=0;i<s1;i++){nn1[i]=nn1[i]+bb1[i];aa1[i]=f1(nn1[i]);}
for(i=0;i<s2;i++){nn2[i]=0.0;for(j=0;j<s1;j++)nn2[i]=nn2[i]+ww2[i][j]*aa1[j];}
for(i=0;i<s2;i++){nn2[i]=nn2[i]+bb2[i];aa2[i]=f1(nn2[i]);}
for(i=0;i<s3;i++){nn3[i]=0.0;for(j=0;j<s2;j++)nn3[i]=nn3[i]+ww3[i][j]*aa2[j];}
for(i=0;i<s3;i++){nn3[i]=nn3[i]+bb3[i];aa3[i]=f3(nn3[i]);ee[i]=tt0[i]-aa3[i];}
//////////////////////////////////////////
err=0.0; for(i=0;i<s3;i++){if(err<(double)fabs(ee[i]))err=(double)fabs(ee[i]);}
//////////////////   Calculating sensitivity ss
for(i=0;i<s3;i++){ss3[i]=-2*df3(nn3[i])*ee[i];}
for(j=0;j<s2;j++){ss2[j]=0;for(i=0;i<s3;i++)ss2[j]=ss2[j]+df1(nn2[j])*ww3[i][j]*ss3[i];}
for(j=0;j<s1;j++){ss1[j]=0;for(i=0;i<s2;i++)ss1[j]=ss1[j]+df1(nn1[j])*ww2[i][j]*ss2[i];}
////////////////// Calculating dww & dbb and and Renew the ww & bb
for(i=0;i<s1;i++)for(j=0;j<s0;j++)dww1[i][j]=gama*dww10[i][j]-(1-gama)*arfa*ss1[i]*aa0[j];
for(i=0;i<s2;i++)for(j=0;j<s1;j++)dww2[i][j]=gama*dww20[i][j]-(1-gama)*arfa*ss2[i]*aa1[j];
for(i=0;i<s3;i++)for(j=0;j<s2;j++)dww3[i][j]=gama*dww30[i][j]-(1-gama)*arfa*ss3[i]*aa2[j];
for(i=0;i<s1;i++)dbb1[i]=gama*dbb10[i]-(1-gama)*arfa*ss1[i];
for(i=0;i<s2;i++)dbb2[i]=gama*dbb20[i]-(1-gama)*arfa*ss2[i];
for(i=0;i<s3;i++)dbb3[i]=gama*dbb30[i]-(1-gama)*arfa*ss3[i];

for(i=0;i<s1;i++)for(j=0;j<s0;j++){ww1[i][j]=ww1[i][j]+dww1[i][j];dww10[i][j]=dww1[i][j];}
for(i=0;i<s2;i++)for(j=0;j<s1;j++){ww2[i][j]=ww2[i][j]+dww2[i][j];dww20[i][j]=dww2[i][j];}
for(i=0;i<s3;i++)for(j=0;j<s2;j++){ww3[i][j]=ww3[i][j]+dww3[i][j];dww30[i][j]=dww3[i][j];}
for(i=0;i<s1;i++){bb1[i]=bb1[i]+dbb1[i];dbb10[i]=dbb1[i];}
for(i=0;i<s2;i++){bb2[i]=bb2[i]+dbb2[i];dbb20[i]=dbb2[i];}
for(i=0;i<s3;i++){bb3[i]=bb3[i]+dbb3[i];dbb30[i]=dbb3[i];}
} k++; ///      end of for(jj=0;jj<n;jj++)

//////////////////////////////////////////
if(k%DL==0){
kk[k_DL]=(double)(k); Err[k_DL]=err;
fprintf(fp1,"\n%d   %lf ",k/DL,err);
//mexPrintf("\nstep=%d  err=%10.7f   arfa=%6.4f   gama=%6.4f   s1=%d s2=%d", k/DL, err, arfa,
gama, s1, s2);
k_DL++;
}    /// end of if(k%DL==0)
if(err<E||k>NStep)break;
} ////   end of while(1)
```

```
////////////////////////// 调用 MATLAB 命令作计算误差图
rP[0]=mxCreateDoubleMATrix(NSS,1,mxREAL);  //////////注：在 C 程序中，矩阵行数与列数须
//////////为 const 类型，否则，内存数据复制时出错，并终止 MATLAB
memcpy(mxGetPr(rP[0]),kk,k_DL*sizeof(double));
lP[0]=mxCreateDoubleMATrix(NSS,1,mxREAL);
memcpy(mxGetPr(lP[0]),Err,k_DL*sizeof(double));
mexCallMATLAB(1,rP,1,lP,"plot");
mxDestroyArray(rP[0]);mxDestroyArray(lP[0]);
//////////////////////////// Output The WW & BB
fprintf(fp1,"\n\n %d %d %d %d  %d",s0,s1,s2,s3,k);
fprintf(fp1,"\n\n");for(j=0;j<s1;j++) {fprintf(fp1,"\n");
for(i=0;i<s0;i++)fprintf(fp1,"    %lf",ww1[j][i]);fprintf(fp1,"     %lf",bb1[j]);}
fprintf(fp1,"\n\n");for(j=0;j<s2;j++) {fprintf(fp1,"\n");
for(i=0;i<s1;i++)fprintf(fp1,"    %lf",ww2[j][i]);fprintf(fp1,"     %lf",bb2[j]);}
fprintf(fp1,"\n\n");for(j=0;j<s3;j++) {fprintf(fp1,"\n");
for(i=0;i<s2;i++)fprintf(fp1,"    %lf",ww3[j][i]);fprintf(fp1,"     %lf",bb3[j]);}
fprintf(fp1,"\n\n");

//////////    Forcasting    //////////////////
////////////////////////////////////////////////////////////////////
fscanf(fp0,"%d",&n);
for(jj=0;jj<n;jj++)for(i=0;i<s0;i++)fscanf(fp0,"%lf",&pp[i][jj]);
for(jj=0;jj<n;jj++){
for(i=0;i<s0;i++)aa0[i]=pp[i][jj];
for(i=0;i<s1;i++){nn1[i]=0.0;for(j=0;j<s0;j++)nn1[i]=nn1[i]+ww1[i][j]*aa0[j];}
for(i=0;i<s1;i++){nn1[i]=nn1[i]+bb1[i];aa1[i]=f1(nn1[i]);}
for(i=0;i<s2;i++){nn2[i]=0.0;for(j=0;j<s1;j++)nn2[i]=nn2[i]+ww2[i][j]*aa1[j];}
for(i=0;i<s2;i++){nn2[i]=nn2[i]+bb2[i];aa2[i]=f1(nn2[i]);}
for(i=0;i<s3;i++){nn3[i]=0.0;for(j=0;j<s2;j++)nn3[i]=nn3[i]+ww3[i][j]*aa2[j];}
for(i=0;i<s3;i++){nn3[i]=nn3[i]+bb3[i];aa3[i]=f3(nn3[i]);
}//////////////////////////////////////////////////////////////////
fprintf(fp1,"\n");for(i=0;i<s3;i++)fprintf(fp1," %lf",aa3[i]);
kk[jj]=pow(10,aa0[0]); Err[jj]=pow(10,aa3[0]);
} /// endof for(jj)
////////////////////////// 调用 MATLAB 命令将预测结果以图形方式显出
rrP[0]=mxCreateDoubleMATrix(70,1,mxREAL);    //////////注：在 C 程序中，矩阵行数与列数须
//////////为 const 类型，否则，内存数据复制时出错，并终止 MATLAB
memcpy(mxGetPr(rrP[0]),kk,n*sizeof(double));
llP[0]=mxCreateDoubleMATrix(70,1,mxREAL);
memcpy(mxGetPr(llP[0]),Err,n*sizeof(double));
mexCallMATLAB(1,rrP,1,llP,"plot");
mexEvalString("xlabel('dat/d');");
mexEvalString("ylabel('settlement/mm');");
mxDestroyArray(rrP[0]);mxDestroyArray(llP[0]);
////////////////////////////////////////////////
fclose(fp0); fclose(fp1);
}
////////////////////用户 C 程序-end //////////////////////////////

///////////////  接口程序 mexFunction////////////////////////////////
```

```
void mexFunction(int nlhs,mxArray *plhs[],int nrhs, mxArray *prhs[])
{
int rP;
rP=mxCalloc(1,sizeof(int));
rP=(int)mxGetScalar(prhs[0]);
if(rP==0) Bp3_C();
else mexErrMsgTxt(" To call User C_program, The paramter must be 0. \n");
}
/////////////////////////////////结束/////////////////////////////////////////////
```

从程序清单中可知，与一般 MEX 文件一样，程序由三部分构成：头文件、用户 C 程序代码、接口程序。该程序的接口函数极其简单，仅从输入参数中读取一个整型数值，以判断程序是否调用用户 C 程序：若为 0，调用用户 C 程序；否则，程序报错并返回 MATLAB；而在用户 C 程序中，仅作了少量修改，如内存分配与释放、程序因各种错误被终止退出等方面。同时，在用户 C 程序中，添加 mexCallMATLAB()函数来调用 MATLAB 中的 plot 命令，并将计算误差和预测结果以图形方式显示出来（图 7-5），实现 MATLAB 与 C 语言混合编程。

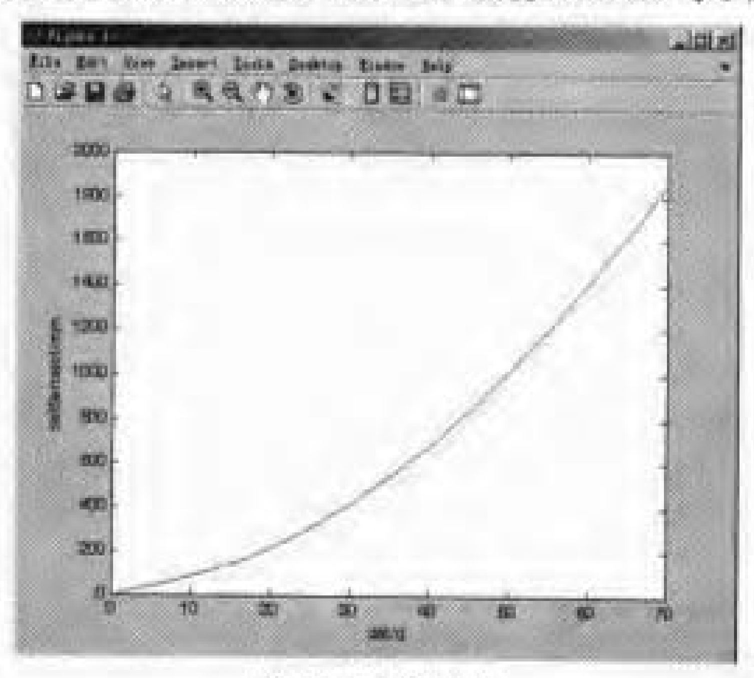


图 7-5　BP 预测曲线

7.3　C 引擎应用程序模式

7.3.1　MATLAB 引擎库函数介绍

引擎应用程序的实质是把MATLAB 作为一个引擎，它允许从用户的C程序调用这个引擎。在运行时，引擎作为一个进程单独运行，C 程序也作为一个进程单独运行，二者之间以 COM（Windows 操作系统）通信机制进行数据交互。C 程序调用 MATLAB 的流程如图 7-6 所示。计算引擎应用程序与图 7.1 所示的 MEX 文件流程正好相反，是以 C 程序为主体的，通过 engine 接口以后台方式启动 MATLAB，并以引擎库函数调用 MATLAB。

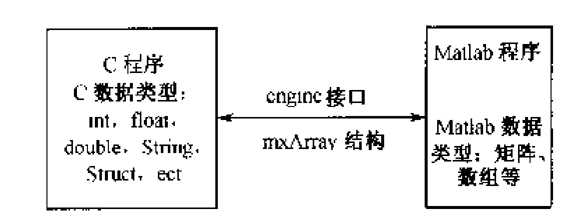


图 7-6　Windows 操作系统下的计算引擎应用程序调用流程

MATLAB 引擎库函数以 eng 为前缀，常用引擎库函数如表 7-1 所示。各引擎函数简介如下。

表 7-1　MATLAB 常用引擎库函数

序　号	函　数	功　能
1	engOpen	启动 MATLAB 引擎
2	engClose	关闭 MATLAB 引擎
3	engGetVariable	从 MATLAB 引擎中获取一个 MATLAB 矩阵
4	engPutVariable	向 MATLAB 引擎发送一个 MATLAB 矩阵
5	engEvalString	执行一个 MATLAB 命令
6	engOutputBuffer	创建一个存储 MATLAB 文本输出的缓冲区
7	engGetVisible	获取 MATLAB 引擎的可见状态
8	engSetVisible	设置 MATLAB 引擎的可见状态

（1）engOpen

Engine *engOpen(const char *startcmd);

参数：startcmd　在 windows 操作系统下，该参数一般设为 NULL。

返回：指向 MATLAB engine 的指针。

说明：由于 MATLAB 引擎在 Windows 操作系统下的通信机制为 COM 的 Server/Client 机制，这需要 MATLAB 在本地计算机上注册 MATLAB engine COM Server。注册方法为：在 Windows 命令行（开始→运行）输入“MATLAB /regserver”即可注册。

（2）engClose

int engClose(Engine *ep);

参数：ep　已打开的指向 MATLAB 引擎的指针。

返回：操作成功返回 0，否则返回 1。

说明：操作失败主要是试图再次关闭早已关闭的 MATLAB 引擎。

（3）engGetVariable

mxArray *engGetVariable(Engine *ep, const char *var_name);

参数：ep　已打开的指向 MATLAB 引擎的指针；

var_name　欲从 MATLAB 引擎空间中获取的变量名。

返回：操作成功返回指向新分配的 mxArray 类型矩阵，否则返回 NULL。

说明：在不需使用该矩阵后，用户程序应释放该矩阵内存空间，以免造成内存泄漏。

（4）engPutVariable

int engPutVariable(Engine *ep, const char *var_name, const mxArray *array_ptr);

参数：ep　已打开的指向 MATLAB 引擎的指针；

var_name　MATLAB 引擎空间中的变量名；

array_ptr 用户程序中已定义的 mxArray 类型指针。

返回：操作成功返回 0，否则返回 1。

说明：当 MATLAB 引擎空间中存在与 var_name 参数同名的 mxArray 类型矩阵时，其数据被覆盖，否则，在 MATLAB 引擎空间中创建以 var_name 参数为名的 mxArray 类型矩阵。

（5）engEvalString

int engEvalString(Engine *ep, const char *string);

参数：ep 已打开的指向 MATLAB 引擎的指针；

string 欲在 MATLAB 中执行的命令。

返回：操作成功返回非零值，否则返回 0。

（6）engOutputBuffer

int engOutputBuffer(Engine *ep, char *p, int n);

参数：ep 已打开的指向 MATLAB 引擎的指针；

p 指向所创建的字符缓冲区指针；

n 缓冲区字节数。

返回：操作成功返回 0，否则返回 1。

说明：engOutputBuffer(ep, NULL, 0)关闭所创建的缓冲区。

（7）engGetVisible

int engGetVisible(Engine *ep, bool *value);

参数：ep 已打开的指向 MATLAB 引擎的指针；

value BOOL 型指针，若 ep 所指的 MATLAB 引擎可见，其值为 1，否则，为 0。

返回：操作成功返回 0，否则返回 1。

（8）engSetVisible

int engSetVisible(Engine *ep, bool value);

参数：ep 已打开的指向 MATLAB 引擎的指针；

value BOOL 型数值，value=1 设置 ep 所指的 MATLAB 引擎可见，value=0 设置 ep 所指的 MATLAB 引擎不可见。

返回：操作成功返回 0，否则返回 1。

说明：若 ep 所指的 MATLAB 引擎可见，则用户可将其用户交互式界面使用。

7.3.2 MATLAB 引擎应用程序示例

本节以一个简单的 C 程序为例，说明 MATLAB 引擎应用开发过程。

```
/////////// 头文件
#include "engine.h"
#include "stdio.h"
#include "conio.h"
////////////////// C 程序
int main()
{
////////////打开引擎
Engine *ep;
if((ep=engOpen(NULL))==NULL){printf("\nCan't Open MATLAB engine\n");exit(-1);}
////////////通过引擎库函数调用 MATLAB 命令
```

```
engEvalString(ep,"x=0:0.1:2*pi;");
engEvalString(ep,"y=sin(x);");
engEvalString(ep,"plot(x,y);");
engEvalString(ep,"xlabel('X');");
engEvalString(ep,"ylabel('Y');");
printf("\n Press anykey to continue\n");
getch();
////////// 关闭引擎
if(engClose(ep)!=0){printf("\nCan't close MATLAB engine normally\n");exit(-1);}
printf("\n MATLAB engine is over\n");
return 0;}
///////////////结束
```

该程序可分四个部分：

（1）头文件，除一般C程序的头文件外，引擎应用程序须包括#include "engine.h"头文件；

（2）在C程序中，调用engOpen()函数打开MATLAB引擎；

（3）在C程序中，通过eng库函数调用MATLAB命令；

（4）在C程序结束之前，调用engClose()函数关闭MATLAB引擎。

与MEX文件编译相似，在编译引擎应用程序之前，需通过mex -setup来指定以不同语言编写的引擎应用程序的编译器。由于各种编译器的设置较为复杂，MATLAB为用户提供了各种编译器的设置参数，称为编译器选项文件，它们存放于%MATLABRoot%\bin\win32\mexopts目录下。在该目录下可以看到，MATLAB为用户提供了从Borland C（从Bcc5.3～Bcc5.6版本）、Fortran（df5.0～df6.6版本）、MSVC（MSVC5.0～MSVC7.1版本）和LCC等十几种编译器的编译选项文件。用户对引擎应用程序的编译，只需以mex -setup指定编译器，并采用与指定编译器相对应的编译选项文件即可。如采用LCC编译器对“engl.c”的编译过程为：

在MATLAB命令行输入

```
>> mex -setup
Please choose your compiler for building external interface (MEX) files:
Would you like mex to locate installed compilers [y]/n?
```

MATLAB提示用户是否重新指定编译器？输入y确定重新指定编译器，MATLAB列出本机上所安装的编译器及其安装路径，并提示用户选取相应的编译器（以下为MATLAB列出的笔者本机上可用的编译器，共3个）。

```
Select a compiler:
[1] Digital Visual Fortran version 6.0 in C:\Program Files\Microsoft Visual Studio
[2] Lcc C version 2.4 in C:\MATLAB7\sys\lcc
[3] Microsoft Visual C/C++ version 6.0 in C:\Program Files\Microsoft Visual Studio
[0] None

Compiler:
```

输入2指定所采用的编译器为Lcc C version 2.4，MATLAB要求用户再次确认所输入的选项。

```
Please verify your choices:
Compiler: Lcc C 2.4
Location: C:\MATLAB7\sys\lcc
Are these correct?([y]/n):
```

输入 y 确认指定 Lcc C 2.4 为指定编译器。

```
Try to update options file: C:\Documents and Settings\Administrator\Application Data\MAThWorks
\MATLAB\R14\mexopts.bat
From template:              C:\MATLAB7\BIN\WIN32\mexopts\lccopts.bat
Done ...
```

至此用户指定应用程序的编译器为 LCC 编译器，相应的编译器选项文件为 lccengmatopts.bat。因此，对"eng1.c"的编译为：

```
>>mex -f c:\MATLAB7\bin\win32\mexopts\lccengmatopts.bat eng1.c
```

若引擎应用程序无错误，编译成功后，在 Current Directory 窗口中可得"eng1.exe"可执行文件。在 Windows 命令行（开始→运行）输入"D:\MATLAB_C\Engine\eng1"（D:\MATLAB_C\Engine 为笔者"eng1.exe"文件存储路径），可以看到，eng1.exe 以后台方式启动 MATLAB（图 7-7 和图 7-8），其命令行提示符为?。用户可在该窗口中执行 MATLAB 各种命令和函数调用。若输入以下命令后，可得图 7-9。

```
? y1=cos(x);
? plot(x,y,x,y1);
? xlabel('X');
? ylabel('Y');
```

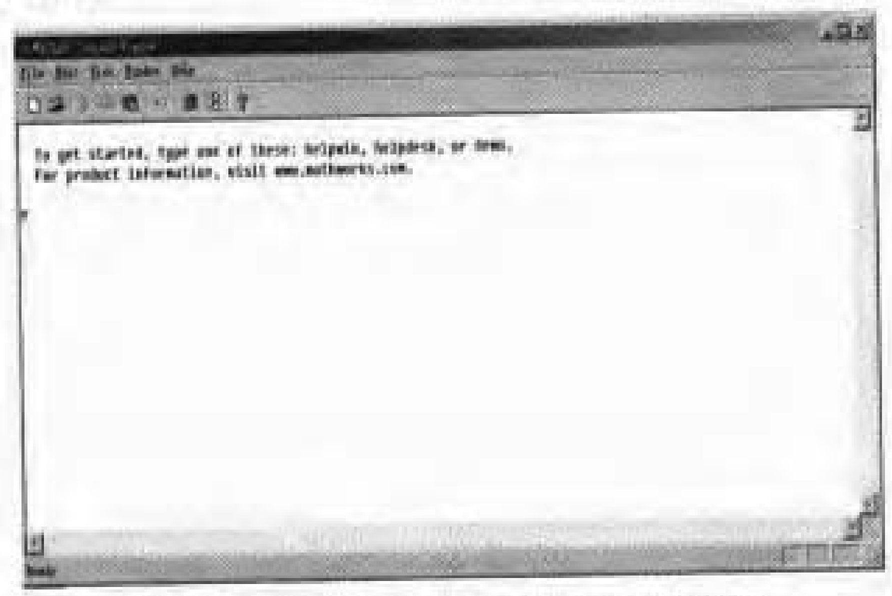


图 7-7　引擎程序以后台方式启动的 MATLAB 命令窗口

7.3.3　在 Visaul C++6.0 中编译、调试引擎应用程序

与其他语言编程一样，用户编写的引擎应用程序很难保证一次成功，往往需要进行大量的调试工作。MATLAB 提供了不同操作系统下引擎应用程序的调试方法。鉴于 Visual C++功能强大的编译和调试功能，用其对 C 语言程序的兼容性，本节在"eng1.c"文件基础上，加入部分引擎库函数得到"eng2.c"，并以 eng2.c 为例说明 Visaul C++6.0 中编译、调试引擎应用程序的步骤。

eng2.c 是通过引擎库函数 engGetVariable()从引擎启动的 MATLAB 空间中将数据输入 C 程序，并在 C 程序中将引擎计算结果输出到文本文件"ng2Out.txt"中。整个引擎应用程序创建过程如下。

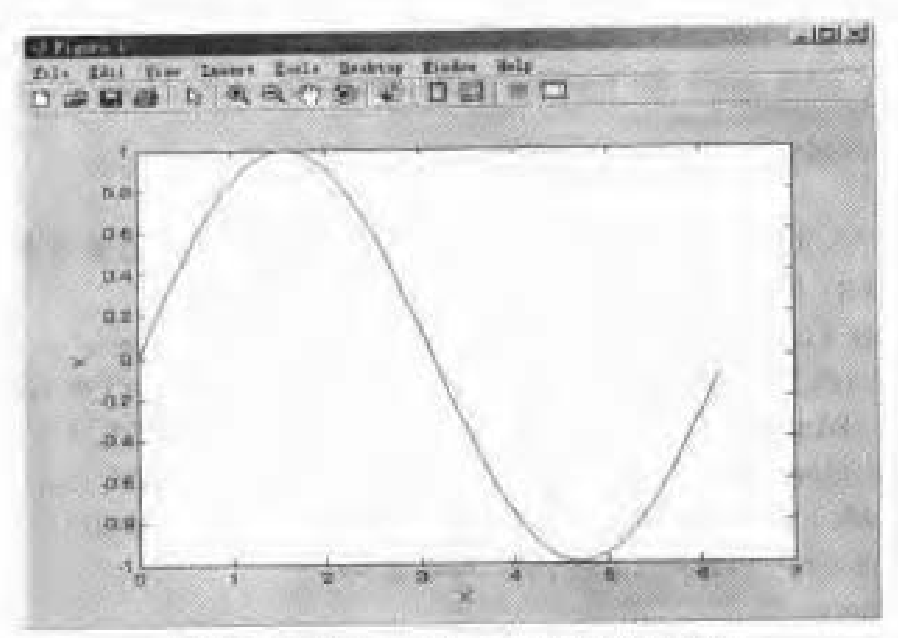


图 7-8 引擎程序（eng1.c）中执行正弦函数结果

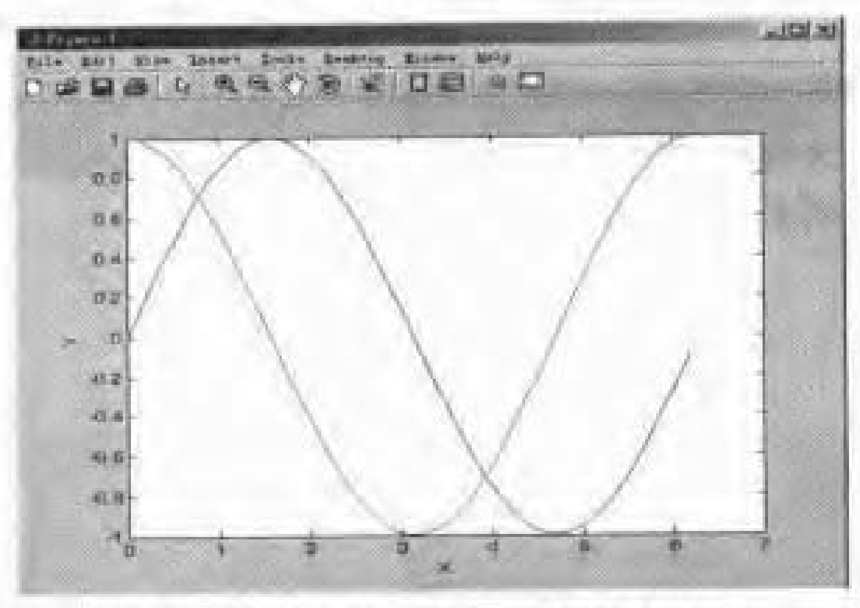


图 7-9 用户在 MATLAB 命令窗口中执行余弦函数结果

（1）启动 Microsoft Visual Studio 后，执行 File→New，在弹出的对话框中选择“Win32 Console Application”作为即将创建的应用程序类型，并在 ProjectName 文本框中输入应用程序工程名，也就是以后生成的可执行程序名，如“Eng2”，点击 OK 确定（图 7-10）。在接下来的对话框中选 An empty project（图 7-11），并以 Finish 完成引擎应用程序工程创建。

（2）在所创建的引擎应用程序工程中，打开已有 C 程序或新建 C 程序（见 Eng2.c 程序清单）。

（3）引擎应用程序编译器路径设置：在 Microsoft Visual Studio 的 Tools→Options 对话框中，点击 Directories 项，在该对话框 Show directories for 中选取 Include files 并在 Directories 中添加“C:\MATLAB7\EXTERN\INCLUDE”（图 7-12），在 Show directories for 中选取 Library files 并在 Directories 中添加“C:\MATLAB7\EXTERN\LIB\WIN32\MICROSOFT\MSVC60”（图 7-13），点击 OK 完成编译路径设置。

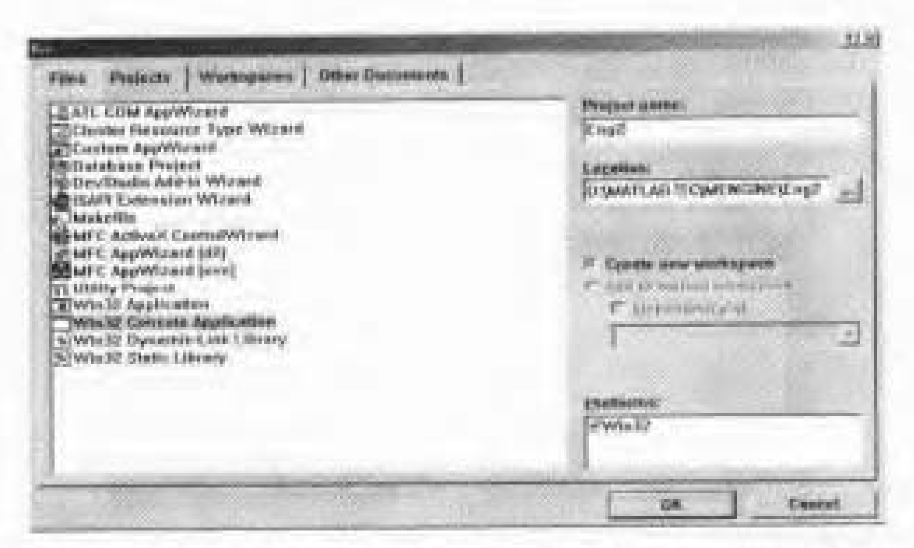


图 7-10　引擎应用程序工程创建（Win32 Console Application 对话框设置）

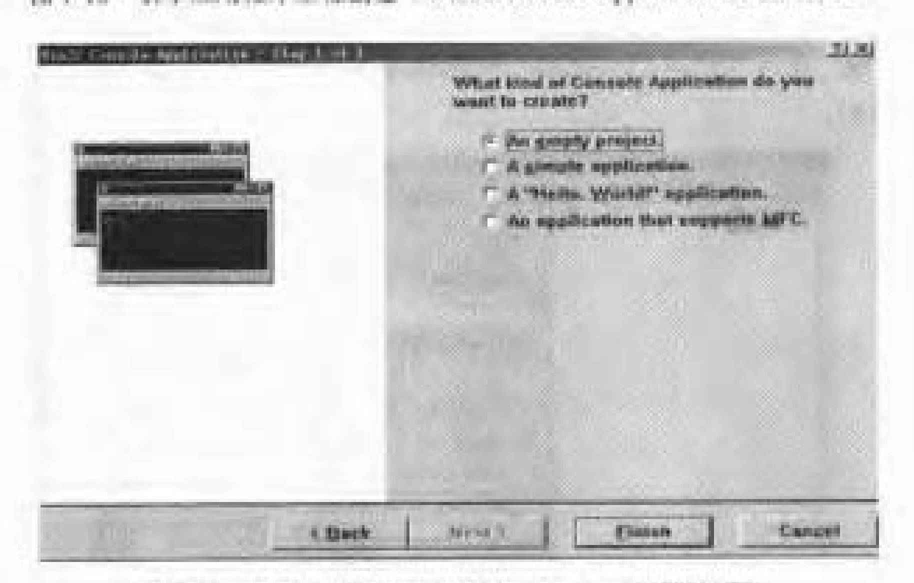


图 7-11　引擎应用程序工程创建（Project 对话框设置）

图 7-12　引擎应用程序的编译路径设置（Include files）

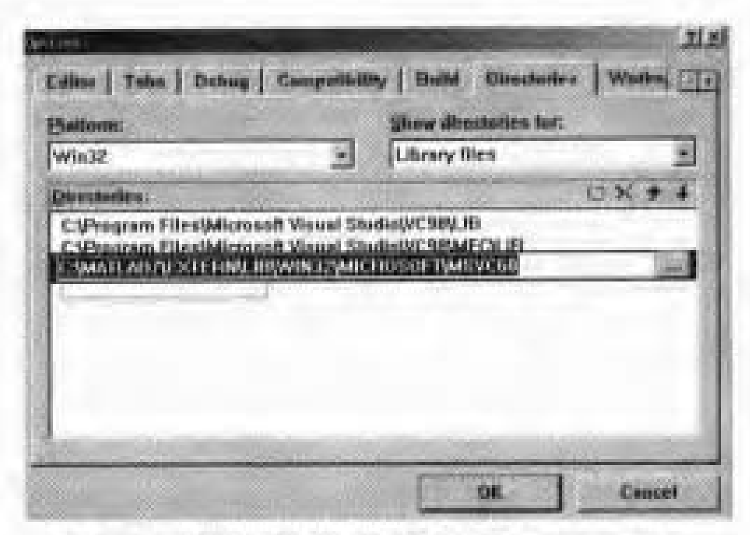


图 7-13　引擎应用程序的编译路径设置（Library files）

（4）引擎应用程序链接路径设置：在 Microsoft Visual Studio 的 Project→Settings 对话框中，点击 Link 项，在该对话框 Object/library modules 中添加 libmx.lib libeng.lib 完成链接路径设置（图 7-14）。

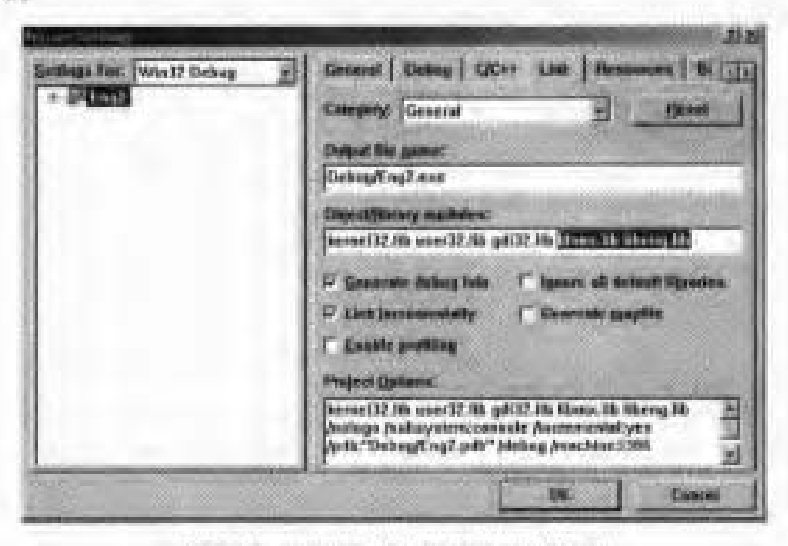


图 7-14　引擎应用程序链接路径设置

至此，已完成了引擎应用程序编译、链接等设置工作，可以对所创建的引擎应用程序进行编译、调试并执行引擎应用程序。

```
///// eng2.c 程序清单////////////////////////////////////////////
//////////头文件
#include "engine.h"
#include "stdio.h"
#include "conio.h"
////////////////C 程序
int main()
{
double *cx,*cy,*cy1;
int n,i;
FILE *fp;
//////////打开引擎
```

```
Engine *ep;
if((ep=engOpen(NULL))==NULL){printf("\nCan't Open MATLAB engine\n");exit(0);}
////////////通过引擎库函数调用 MATLAB 命令
engEvalString(ep,"x=0:0.1:2*pi;");
engEvalString(ep,"y=sin(x);");
engEvalString(ep,"y1=cos(x);");
engEvalString(ep,"plot(x,y,x,y1);");
engEvalString(ep,"xlabel('X');");
engEvalString(ep,"ylabel('Y');");
//////////// 通过引擎库函数将后台 MATLAB 空间中的数据传入 C 程序
n=mxGetN(engGetVariable(ep,"x"));
cx=mxGetPr(engGetVariable(ep,"x"));
cy=mxGetPr(engGetVariable(ep,"y"));
cy1=mxGetPr(engGetVariable(ep,"y1"));

printf("\n Press anykey to continue\n"); getch();
//////////关闭引擎
if(engClose(ep)!=0){printf("\nCan't close MATLAB engine normally\n");exit(-1);}
printf("\n MATLAB engine is over\n");
/////////将 MATLAB 引擎计算结果输出
if((fp=fopen("eng2Out.txt","w"))==NULL){printf(" Can't open OutPutfile\n");exit(0);}
for(i=0;i<n;i++)fprintf(fp," %3d  %12.4lf    %12.4lf  %12.4lf \n",i,cx[i],cy[i],cy1[i]);
/////////释放内存
mxFree(cx); mxFree(cy);
fclose(fp);
return 0;}
///////////////结束
```

7.3.4 MATLAB 引擎应用程序实例开发

本节通过引擎库函数调用 MATLAB 神经网络工具，来求解 7.2.6 中“BP3.c”两隐含层神经网络问题，以进一步说明 MATLAB 引擎应用开发过程。

1．训练函数简介

该两隐含层的神经网络问题求解，算法采用有动量的梯度下降法。在 MATLAB 神经网络工具箱中，提供了如下两种训练方法。

（1）先构建网络并设定训练用算法，然后再应用最基本的批处理训练函数 train()。

net=newff(PR,[s1,s2,..,sn],{TF1,TF2,…,TFn},BTF,BLF,PF);

参数：PR　输入样本极值，一般由 minmax()函数取得；

[si]　第 i 层神经元数；

TFi　第 i 层传递函数，常用的有'logsig'、'tansig' 和 'purelin'三种，默认值'tansig'；

BTF　训练函数，常用的有：'traingd'、'traingdm'、'traingdx'、'trainrp'等，默认值'traingdx'；

BLF　阈值和权重学习函数，默认值'learngdm'；

PF　演示函数，默认值'mse'。

[net,tr,Y,E,Pf,Af]=train(NET,P,T,Pi,Ai,VV,TV)；或 [net,tr]=train(NET,P,T)；

参数：NET　网络结构；

P　网络输入，Ni×Ts 的矩阵（Ni 为输入向量数目(Ni=net.numInputs)；Ts 为总

时步数）；

T　训练目标，Nt×Ts矩阵，默认值为[0]（Nt为目标向量数目(Nt＝net.numTargets), Ts同上）；

Pi　初始输入延迟条件，默认值为[0]；

Ai　初始延迟条件，默认值为[0]；

VV　有效向量，默认值为[]；

TV　检测向量，默认值为[]。

返回：net　网络结构；

tr　训练记录（回显及演示）；

Y　网络输出；

E　网络错误；

Pf　输入延迟条件；

Af　延迟条件。

（2）直接用traingdm()函数训练。

[net,tr,Ac,El] = traingdm(net,Pd,Tl,Ai,Q,TS,VV,TV)

参数与返回说明见方法（1）。

训练参数设置说明如表7-2所示。

表7-2　Traingdm训练参数设置说明

参　数	说　明
net.trainParam.epochs	最大训练步数，默认值为10
net.trainParam.goal	训练要求精度，默认值为0
net.trainParam.lr	学习率，默认值为0.01
net.trainParam.max_fail	最大失败次数，默认值为5
net.trainParam.mc	动量因子，默认值为0.9
net.trainParam.min_grad	最小梯度要求，默认值为1e-10
net.trainParam.show	显示训练迭代过程，默认值为25，NaN不显示
net.trainParam.time	最大训练时间，默认值为inf

训练终止条件如下（只满足其中的一个条件，训练结束）：

- 训练时步数达到预定值；
- 训练耗时达到预定值；
- 训练误差小于预定值；
- 下降梯度小于预定值；
- 训练失败次数达预定值。

2. 引擎应用程序清单

该引擎应用程序的网络结构模型及数学模型见7.2.6节。引擎应用程序清单如下。

```
/////////// 头文件
#include "engine.h"
#include "stdio.h"
#include "conio.h"
#include "string.h"
////////////////// C程序
```

```
int main()
{

FILE *fp0,*fp1;
int i,j,s0,s1,s2,s3,n,n0,ts;
char *DataFile,*Fss,*Fout;
double lr,mc;
mxArray *S,*p,*t,*a,*Lr,*Mc,*Ts;
Engine *ep;

DataFile="J6.txt"; ////////////////////////////////////输入文件名
if((fp0=fopen(DataFile,"r"))==NULL){printf(" Can't open OutPutfile\n");exit(0);}
Fout=strdup(DataFile);Fss="-r.txt";strcat(Fout,Fss);
if((fp1=fopen(Fout,"w"))==NULL){printf(" Can't open OutPutfile\n");exit(0);}

S=mxCreateDoubleMATrix(1,3,mxREAL);
Lr=mxCreateDoubleMATrix(1,1,mxREAL);
Mc=mxCreateDoubleMATrix(1,1,mxREAL);
Ts=mxCreateDoubleMATrix(1,1,mxREAL);
fscanf(fp0,"%d%d%d%d%d%d%lf%lf%d",&n,&n0,&s0,&s1,&s2,&s3,&lr,&mc,&ts);
//////////////////将网络控制参数赋给 mxArray 矩阵：S
*(mxGetPr(S)+0)=(double)s1; *(mxGetPr(S)+1)=(double)s2;
*(mxGetPr(S)+2)=(double)s3; //*(mxGetPr(S)+3)=(double)s3;
*(mxGetPr(Lr))=(double)lr;   *(mxGetPr(Mc))=(double)mc;   *(mxGetPr(Ts))=(double)ts;
p=mxCreateDoubleMATrix(s0,n,mxREAL);
t=mxCreateDoubleMATrix(s0,n,mxREAL);

///////////////////输入样本与目标向量并赋给 mxArray 矩阵：p&t
for(j=0;j<n;j++){
for(i=0;i<s0;i++)fscanf(fp0,"%lf",(mxGetPr(p)+j*s0+i));
for(i=0;i<s3;i++)fscanf(fp0,"%lf",(mxGetPr(t)+j*s0+i));
}    n=n0;
/////////////////////////输入预测
fscanf(fp0,"%d",&n);a=mxCreateDoubleMATrix(s3,n,mxREAL);
for(j=0;j<n;j++)for(i=0;i<s0;i++)fscanf(fp0,"%lf",(mxGetPr(a)+j*s3+i));

//////////////打开引擎
if((ep=engOpen(NULL))==NULL){printf("\nCan't Open MATLAB engine\n");exit(0);}
///////////// 通过引擎库函数调用 MATLAB 命令
engPutVariable(ep,"ep_p",p);   ////将学习样本置入引擎空间
engPutVariable(ep,"ep_t",t);   ////将目标样本置入引擎空间
engPutVariable(ep,"ep_a",a);   ////将预测样本置入引擎空间
engPutVariable(ep,"ep_S",S);   ////将相关网络参数置入引擎空间
engPutVariable(ep,"ep_Lr",Lr);
engPutVariable(ep,"ep_Mc",Mc);
engPutVariable(ep,"ep_Ts",Ts);

engEvalString(ep,"net=newff(minmax(ep_p),ep_S,{'logsig','logsig','purelin'},'traingdm');");
///构造 BP 网络
engEvalString(ep,"net.trainParam.show=200;");      ///回显一次所经过的计算时步
engEvalString(ep,"net.trainParam.lr=ep_Lr;");      ///学习率
engEvalString(ep,"net.trainParam.mc=ep_Mc;");      ///动量项
```

```
engEvalString(ep,"net.trainParam.epochs=ep_Ts;");    //预定时步
engEvalString(ep,"net.trainParam.goal=1e-5;");    ///训练要求精度
engEvalString(ep,"net.trainParam.time=inf;");    ///无训练时间限制
engEvalString(ep,"[net,tr]=train(net,ep_p,ep_t);");  ///训练网络

engEvalString(ep,"ep_b=sim(net,ep_a);");    ////////////样本预测

printf("\nPress anykey to plot\n"); getch(); ////////////作预测曲线
engEvalString(ep,"ep_x=(10.0).^ep_a; ep_y=(10.0).^ep_b;");
engEvalString(ep,"plot(ep_x,ep_y);");
engEvalString(ep,"xlabel('Date/d');");
engEvalString(ep,"ylabel('Displacement/mm');");

/////////////////////////////预测结果返回
printf("\nPress anykey to return and engClose \n"); getch();
for(i=0;i<n;i++)fprintf(fp1," %3d %12.4lf    %12.4lf  \n",
i,*(mxGetPr(engGetVariable(ep,"ep_x"))+i),*(mxGetPr(engGetVariable(ep,"ep_y"))+i));

//////////  关闭引擎
if(engClose(ep)!=0){printf("\nCan't close MATLAB engine normally\n");exit(-1);}
printf("\n MATLAB engine is over\n");
///////// 将 MATLAB 引擎计算结果输出
///////// 释放内存
mxDestroyArray(p);  mxDestroyArray(t);   mxDestroyArray(a);   mxDestroyArray(S);
mxDestroyArray(Lr); mxDestroyArray(Mc);  mxDestroyArray(Ts);
fclose(fp0); fclose(fp1);    ///  关闭文件
return 0;}
///////////////结束//////////////////////////////////////////////////////////////////////////
```

7.4 MAT 文件模式

7.4.1 MAT 文件格式介绍

MAT 文件是 MATLAB 提供的一种特殊的数据文件格式——MAT 格式，它是一种二进制格式文件，扩展名为.mat，这些*.mat 文件头带有设备签名，因而可以独立于操作平台，从而实现不同操作平台间的数据传送。

MAT 文件格式如图 7-15 所示，包括文件头和数据。其中文件头（VariableInfo）部分主要包括文件描述、版本号及标识等，共占 128 字节；文件数据（Variable）包括数据类型、数据长度和数据三部分。因此，C 程序对 MAT 文件的读取函数（表 7-3），可相应地分为两种，一是读取 mxArray 变量的头部信息和变量数据；二是仅读取 mxArray 变量的头部信息。

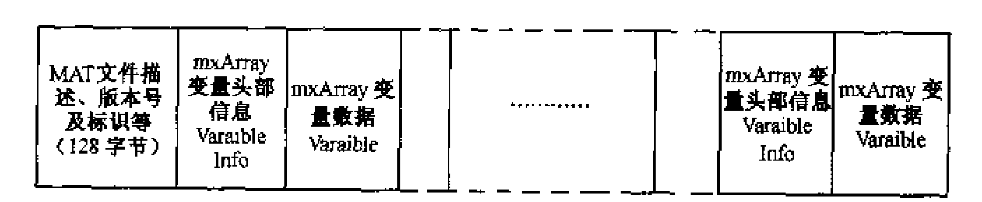


图 7-15 MAT 文件存储格式

表 7-3　C 程序对 MAT 文件的读取函数

序　号	MAT 函数	功　　能
1	matOpen	打开一个 MAT 文件
2	matClose	关闭一个 MAT 文件
3	matGetDir	从 MAT 文件中获取 mxArray 矩阵列表
4	matGetFp	获取一个 ANSI C 文件指针给 MAT 文件
5	matGetVariable	从 MAT 文件中装载 mxArray 矩阵
6	matGetVariableInfo	从 MAT 文件中仅装载 mxArray 矩阵的头部信息
7	matDeleteVariable	从 MAT 文件中删除一个 mxArray 矩阵
8	matGetNextVariable	从 MAT 文件中装载下一个 mxArray 矩阵
9	matGetNextVariableInfo	从 MAT 文件中仅装载下一个 mxArray 矩阵的头部信息
10	matPutVariable	将一个 mxArray 矩阵写入 MAT 文件
11	matPutVariableAsGlobal	将 mxArray 矩阵作为全局空间的变量写入 MAT 文件

（1）matOpen

MATFile *matOpen(const char *filename, const char *mode);

参数：filename　欲打开的 MAT 文件名；

mode　欲打开 MAT 文件的读写模式（表 7-4）。

表 7-4　MAT 文件主要打开模式

参数（mode）	说　　明
R	Read 模式，即只读模式
W	Write 模式，即只写模式
U	Update 模式，即可读取、写入模式
WZ	Write&Zip 模式，即压缩数据格式写入模式

返回：操作成功返回一个指向 MAT 文件的指针，否则，返回 NULL。

（2）matClose

int matClose(MATFile *mfp);

参数：mfp　指向 MATFile 文件的指针。

返回：操作成功返回 0，否则，返回 EOF。

（3）matGetDir

char **matGetDir(MATFile *mfp, int *num);

参数：mfp　指向 MATFile 文件的指针；

num　mxArray 矩阵在 MAT 文件中的地址。

返回：操作成功，返回 mxArray 在 MAT 文件中的地址，并将 mxArray 长度值赋给 num；若 MAT 文件中无 mxArray，返回 0；操作失败，返回 NULL，num 被赋为负值。

（4）matGetFp

FILE *matGetFp(MATFile *mfp);

参数：mfp　指向 MATFile 文件的指针。

返回：操作成功，返回一个 FILE 类型的指向 MAT 文件的指针；否则，返回 NULL。

（5）matGetVariable

mxArray *matGetVariable(MATFile *mfp, const char *name);

参数：mfp　指向 MATFile 文件的指针；

　　name　从 MAT 文件中取出的 mxArray 变量名。

返回：操作成功，将 MAT 文件中的 mxArray 变量复制出并赋给分配地址的变量，该变量地址指针即为函数返回值；否则，返回 NULL。

（6）matGetVariableInfo

```
mxArray *matGetVariableInfo(MATFile *mfp, const char *name);
```

参数：mfp　指向 MATFile 文件的指针；

　　name　从 MAT 文件中取出的 mxArray 变量名。

说明：仅读取 MAT 文件中 mxArray 变量的头部信息。

（7）matDeleteVariable

```
int matDeleteVariable(MATFile *mfp, const char *name);
```

参数：mfp　指向 MATFile 文件的指针；

　　name　从 MAT 文件中删除的 mxArray 变量名。

返回：操作成功，返回 0；否则，返回非 0 值。

（8）matGetNextVariable

```
mxArray *matGetNextVariable(MATFile *mfp, const char *name);
```

参数：mfp　指向 MATFile 文件的指针；

　　name　从 MAT 文件中取出的 mxArray 变量名。

返回：操作成功，将 MAT 文件中的 mxArray 变量复制出并赋给分配地址的变量，该变量地址指针即为函数返回值；否则，返回 NULL。

说明：与 matGetVariable()函数不一样的是，matGetNextVariable 允许用户将 MAT 文件中的 mxArray 变量逐个读出。

（9）matGetNextVariableInfo

```
mxArray *matGetNextVariableInfo(MATFile *mfp, const char *name);
```

参数：mfp　指向 MATFile 文件的指针；

　　name　从 MAT 文件中取出的 mxArray 变量名。

说明：将 MAT 文件中 mxArray 变量的头部信息逐个读出。

（10）matPutVariable

```
int matPutVariable(MATFile *mfp, const char *name, const mxArray *mp);
```

参数：mfp　指向 MATFile 文件的指针；

　　name　写入 MAT 文件的 mxArray 变量名；

　　mp　写入 MAT 文件的 mxArray 变量指针。

返回：操作成功，返回 0；否则，返回非 0 值。

说明：若 MAT 文件中不存在 name，将 mp 追加于 mat 文件尾；若 name 已存在于 MAT 文件中，则将其覆盖。

（11）matPutVariableAsGlobal

```
int matPutVariableAsGlobal(MATFile *mfp, const char *name, const mxArray *mp);
```

参数：mfp　指向 MATFile 文件的指针；

　　name　写入 MAT 文件的 mxArray 变量名；

　　mp　写入 MAT 文件的 mxArray 变量指针。

返回：操作成功，返回 0；否则，返回非 0 值。

说明：与 matPutVariable()函数不同的是，matPutVariableAsGlobal()将 mxArray 变量作为全局空间的变量写入 MAT 文件，当加载该变量时，也被当作全局空间中的变量加载。

7.4.2 MAT 文件示例

MAT 文件可由 MATLAB 命令 save 直接生成，也可在 MEX 文件或 engine 应用程序中，由用户 C 程序生成。本节以 MEX 应用程序“MAT1.c”为例，说明 MAT 文件的创建、打开与数据输入输出。该程序可分为如下 3 部分。

（1）头文件：主要是使用#include 指令将 mat.h 和 mex.h 文件包含于“MAT1.c”文件中；

（2）用户 C 程序：主要完成 mxArray 数据对象的赋值和写入 mat 文件操作；

（3）MEX 接口程序：用户 C 程序与 MATLAB 间数据变换，即将 MATLAB 中文件外送入 MEX 文件中。

“MAT1.c”程序清单如下：

```
/////////////// Head //////////////////////////////////////
#include "mex.h"
#include "mat.h"
///////////////// C Program //////////////////////////////////////
void OutPut(MATFile *fp)
{   mxArray *m_str,*m_P;   int i,j;
    double rx[]={1,2,3,4,5,6};        double ix[]={6,5,4,3,2,1};
    double ry[]={1,1,1,1,1,1};        double iy[]={6,6,6,6,6,6};
    m_P=mxCreateDoubleMATrix(2,6,mxCOMPLEX);///// 创建 mxArray 对象
    for(i=0;i<1;i++)for(j=0;j<6;j++)
    {*(mxGetPr(m_P)+j*2+i)=rx[j];*(mxGetPi(m_P)+j*2+i)=ix[j];}
    for(i=1;i<2;i++)for(j=0;j<6;j++)
    {*(mxGetPr(m_P)+j*2+i)=ry[j];*(mxGetPi(m_P)+j*2+i)=iy[j];}
    matPutVariable(fp,"m_P",m_P);    /////将 mxArray 对象写入 mat
    m_str=mxCreateString(" mat File");    //// 创建 mxArray 对象
    matPutVariable(fp,"m_str",m_str);    /////将 mxArray 对象写入 mat
    mxDestroyArray(m_str);mxDestroyArray(m_P); //释放内存
}

/////////////// mexFunction ////////////////////////////////
void mexFunction(int nlhs,mxArray *plhs[],int nrhs, mxArray *prhs[])
{    MATFile *fp; int buf;
mxChar *FileName;
/////////////// Get data from prhs[]
if(nrhs!=1||mxGetClassID(prhs[0])!=mxCHAR_CLASS)
mexErrMsgTxt("Please Input a string to be the MATFile name\n");
buf=(mxGetM(prhs[0])*mxGetN(prhs[0]))+1;
FileName=mxCalloc(buf,sizeof(char));
mxGetString(prhs[0],FileName,buf);
//////////////////////////////打开 mat 文件
if((fp=matOpen(FileName,"w"))==NULL)mexErrMsgTxt("Can't Open MATFile");
OutPut(fp);   ///调用 C 程序，并将数据写入 mat 文件
mexPrintf("\n The '%s.mat' mat_File is created successfully!\n",FileName);
mxFree(FileName); //////释放内存
```

```
matClose(fp); /////关闭 mat 文件
}
////////////////结束////////////////////////////////////////////////
```

从“mat1.c”程序清单中可知，mat 文件数据输入输出主要是 mxArray 类型数据的操作。因此，向 mat 文件中输入输出数据，主要是 mat 库函数对 mxArray 数据对象的创建、赋值、读取、传递与写入等操作。在 MATLAB 命令窗口中将“MAT1.c”编译为 MEX 动态链接子程序，其调用与加载过程如下：

```
>> mex mat1.c
>> mat1('aaa')
   The 'aaa.mat' mat_File is created successfully!
>> load aaa -mat
>> whos
  Name          Size                    Bytes  Class
   m_P          2x6                       192  double array (complex)
   m_str        1x9                        18  char array
Grand total is 21 elements using 210 bytes

>> m_P
m_P =
1.0000+6.0000i  2.0000+5.0000i  3.0000+4.0000i  4.0000+3.0000i  5.0000+2.0000i  6.0000+1.0000i
1.0000+6.0000i  1.0000+6.0000i  1.0000+6.0000i  1.0000+6.0000i  1.0000+6.0000i  1.0000+6.0000i
>> m_str
m_str =
mat File
```

第8章　MATLAB与Visual Basic接口

MATLAB与Visual Basic接口，有使用一般动态链接库、DDE、OLE、ActiveX和COM组件等多种方法。

使用一般动态链接库技术，通常是用MATLAB编译器将M文件编译成动态链接库文件，然后在VB环境中声明库以后调用库中的函数，实现一定的功能。因为数据类型不兼容等原因，当库函数有输入参数和返回值时很容易出错。此时一般采用外部文件数据传输的办法来解决，但数据很大时效率受损。而DDE技术自OLE、ActiveX和COM组件等技术出来以后，也显示出劣势，用得不是很多了。

所以，本章主要从OLE技术开始介绍，比较详细地介绍自OLE技术以来的主要接口技术。重点推荐基于ActiveX和COM组件的接口技术。学习这些内容，需要读者具有OLE自动化编程、ActiveX编程、面向对象编程和COM组件编程方面的基础。

8.1　基于OLE的接口实现

MATLAB支持OLE自动化服务器兼容。自动化是允许一个应用程序或组件控制另一个应用程序或组件的协议。这样，MATLAB可以被任何支持该协议的Windows程序启动和控制，这样的程序包括Visual C++、Visual Basic、Excel、Access和Project等。使用自动化，可以运行MATLAB命令，并可以从MATLAB获取数组数据或把数组数据从MATLAB中输出。

8.1.1　实现OLE自动化

采用OLE自动化时，自动化方法的变量和返回值的数据类型为自动化数据类型，它由自动化协议定义，与具体的语言无关。例如，BSTR是一个宽字符字符串类型，作为一个自动化类型，它与Visual Basic中保存字符串的类型相同。尽管声明和操作的细节由控制器来指定，但任何OLE适用的控制器都应该支持这些数据类型。

注册信息中MATLAB对象的名称为MATLAB.Application，没有大小写区分。调用MATLAB服务器的确切方式取决于你选择的控制器，但所有控制器都需要这个名称来确认服务器。实际操作时需要首先创建MATLAB对象的实例，然后调用该实例对象的方法实现指定功能。往往使用CreateObject函数来创建MATLAB对象的实例，该函数的调用格式一般为：

```
Dim objMATLAB As Object
Set objMATLAB = CreateObject("MATLAB.Application")
```

MATLAB自动化的Execute方法可以在Excel、VB或任何支持VBA的程序中使用。Execute方法把一个命令字符串作为变量并返回一个字符串，其调用格式如下。

如采MATLAB为客户程序：

```
result = h.Execute('command')
result = Execute(h, 'command')
result = invoke(h, 'Execute', 'command')
```

如果 Visual Basic 为客户程序：

[out] BSTR result = Execute([in] BSTR "command")

Execute 函数在句柄h表示的自动化服务器中执行字符串 command 指定的 MATLAB 语句，服务器将输出返回到字符串 result 中，该字符串还可以包含任何警告或出错信息。

下面的例子在服务器中执行 MATLAB 的 version 函数，并将输出返回到 MATLAB 客户程序中。

MATLAB 作为客户程序时，使用下面的命令行：

```
h = actxserver('matlab.application');

server_version = h.Execute('version')
server_version =
ans =
7.0.1.24704 (R14) Service Pack 1
```

Visual Basic 作为客户程序时，使用类似下面的语句行：

```
Dim Matlab As Object
Dim server_version As String

Set Matlab = CreateObject("matlab.application")
server_version = Matlab.Execute("version")
```

下面的例子用 Visual Basic 调用 MATLAB 进行多项式拟合计算和绘图。首先在 Visual Basic 环境中设计窗体，如图 8-1 所示。窗体中各控件的属性设置如表 8-1 中所示。



图 8-1 设计对话框

表 8-1 各控件的属性设置

对象类型	Name 属性	Caption 属性或 Text 属性	其他属性
Form	frmVBOLE	Caption="VB 调用 MATLAB—多项式拟合"	
TextBox	Text1	Text="8"	
	Text2	Text="1"	
	Text3	Text="1 2 3 4 5 6 7 8"	Mutiline=True ScrolBars=2
	Text4	Text="12 24 43 57 70 91 110 200"	Mutiline=True ScrolBars=2
	Text5	Text=""	Mutiline=True ScrolBars=2
Label	Label1	数据组数	
	Label2	X	
	Label3	Y	
	Label4	多项式系数和常数项	
	Label5	多项式的阶次	
CommandButton	Command1	计算	
	Command2	绘图	
	Command3	退出	

在窗体中输入下面的代码，利用 VB 前端界面中输入的数据参数，调用 MATLAB 进行计算和绘图。

```
Option Explicit

'将 MATLAB 实例对象定义为公共变量
Public objMATLAB As Object

'定义一个实现计算或绘图的过程
Private Sub ComputeorPlot(CorP As Boolean)
  Dim intNum As Integer
  Dim intLevel As Integer
  Dim x(1 To 100) As Double
  Dim y(1 To 100) As Double
  Dim strModel As String
  Dim i As Integer
  Dim strCommand As String

  intNum = Val(Text1.Text)
  intLevel = Val(Text2.Text)
  Open App.Path + "/datX" For Output As 1
    Print #1, Text3
  Close 1
  Open App.Path + "/datY" For Output As 1
    Print #1, Text4
  Close 1

  Open App.Path + "/datX" For Input As 1
    For i = 1 To intNum
        Input #1, x(i)
    Next i
  Close 1
  Open App.Path + "/datY" For Input As 1
    For i = 1 To intNum
        Input #1, y(i)
    Next i
  Close 1

  '定义在 MATLAB 中要执行的命令
  strCommand = "n=" & Str(intLevel) & ";x=["
  For i = 1 To intNum
    strCommand = strCommand & Str(x(i)) & " "
  Next i
  strCommand = strCommand & "];y=["
  For i = 1 To intNum
    strCommand = strCommand & Str(y(i)) & " "
  Next i
  strCommand = strCommand & "];"
  If CorP Then
    strCommand = strCommand & "polyfit(x,y,n)"
```

```
        Text5 = objMATLAB.execute(strCommand)
    Else
        strCommand = strCommand & "plot(x,y)"
        objMATLAB.execute (strCommand)
    End If

End Sub

'进行多项式拟合
Private Sub Command1_Click()
    Dim bolCorP As Boolean
    bolCorP = True
    Call ComputeorPlot(bolCorP)
End Sub

'进行绘图
Private Sub Command2_Click()
    Dim bolCorP As Boolean
    bolCorP = False
    Call ComputeorPlot(bolCorP)
End Sub

'退出程序
Private Sub Command3_Click()
    Set objMATLAB = Nothing
    Unload frmVBOLE
End Sub

Private Sub Form_initialize()
    '创建 MATLAB 的实例
    Set objMATLAB = CreateObject("matlab.application")
End Sub
```

运行程序，在各窗口中输入参数，单击“计算”按钮和“绘图”按钮，分别实现计算和绘图。如图 8-2 所示，利用缺省时给定的一套数据和参数，单击“计算”按钮时显示多项式的系数和常数项，于是可以得到多项式拟合模型。

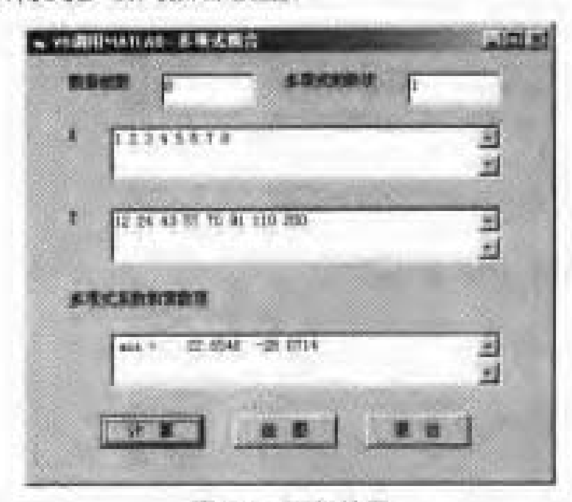


图 8-2 运行结果

单击“绘图”按钮，在 MATLAB 图形窗口中绘制给定数据的线形图，如图 8-3 所示。

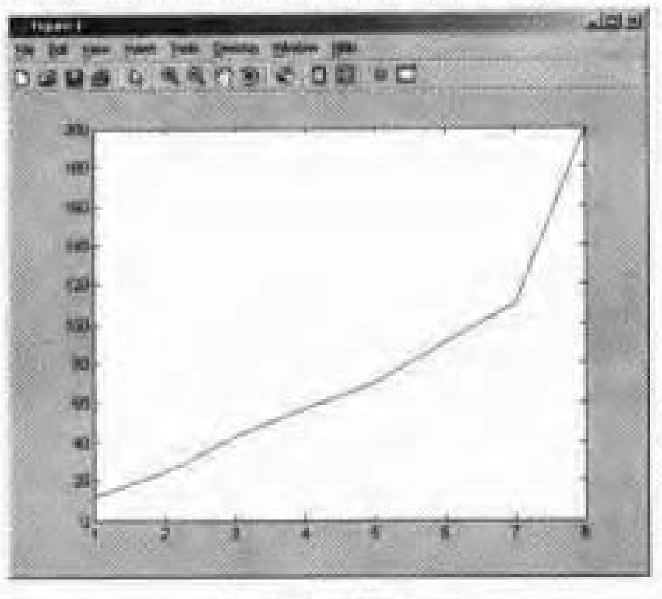


图 8-3　线形图

8.1.2　传递矩阵数据

MATLAB 擅长矩阵计算，使用 GetFullMatrix 和 PutFullMatrix 函数可以在 MATLAB 和 VB 之间传递矩阵数据。

1．GetFullMatrix 函数

该函数从服务器获取矩阵，其调用格式如下。

如果 MATLAB 作为客户程序：

```
[xreal ximag] = h.GetFullMatrix('varname', 'workspace', zreal, zimag)
[xreal ximag] = GetFullMatrix(h, 'varname', 'workspace', zreal, zimag)
[xreal ximag] = invoke(h,'GetFullMatrix', 'varname', 'workspace', zreal, zimag)
```

如果 Visual Basic 作为客户程序：

```
void GetFullMatrix([in] BSTR varname, [in] BSTR workspace,
    [in, out] SAFEARRAY(double)* xreal,
    [in, out] SAFEARRAY(double)* ximag);
```

GetFullMatrix 函数从句柄 h 表示的服务器程序的工作空间 workspace 中获取保存在变量 varname 中的矩阵，并将实部保存在 xreal 中，将虚部保存在 ximag 中。workspace 参数的值可以取 base 或 global。zreal 和 zimag（xreal 和 ximag）参数为大小相同的矩阵。

注意：*如果试图将 GetFullMatrix 函数的输出显示到客户窗口，必须指定 xreal 和/或 ximag 输出变量。使用第一种语法格式时，服务器函数名，例如 GetFullMatrix 是有大小写区分的。对于 MATLAB 客户程序，上面 3 种语法格式在操作效果上没有区别。对于 VB 脚本客户程序，使用 GetWorkspaceData 和 PutWorkspaceData 函数与 MATLAB 工作空间传递数值数据，这些函数使用 variant 数据类型取代 safearray 数据类型，因为 VB 脚本语言不支持 safearray 数据类型。*

下面的例子将一个 5×5 的实数矩阵赋给服务器基本工作空间中的变量 M，然后用 GetFullMatrix 函数将它读回。

MATLAB 作为客户程序时，使用下面的命令行：

```
h = actxserver('matlab.application');
h.PutFullMatrix('M', 'base', rand(5), zeros(5));

MReal = h.GetFullMatrix('M', 'base', zeros(5), zeros(5))
MReal =
    0.9501    0.7621    0.6154    0.4057    0.0579
    0.2311    0.4565    0.7919    0.9355    0.3529
    0.6068    0.0185    0.9218    0.9169    0.8132
    0.4860    0.8214    0.7382    0.4103    0.0099
    0.8913    0.4447    0.1763    0.8936    0.1389
```

Visual Basic 作为客户程序时，使用类似下面的语句行：

```
Dim MatLab As Object
Dim Result As String
Dim XReal(1, 3) As Double
Dim XImag() As Double
Dim RealValue As Double
Dim i, j As Integer
Set Matlab = CreateObject("matlab.application")
Result = MatLab.Execute("M = rand(5);")
Call MatLab.GetFullMatrix("M", "base", XReal, XImag)
```

2. PutFullMatrix 函数

该函数将矩阵保存到服务器程序中，调用格式如下。

如果 MATLAB 作为客户程序：

```
h.PutFullMatrix('varname', 'workspace', xreal, ximag)
PutFullMatrix(h, 'varname', 'workspace', xreal, ximag)
invoke(h, 'PutFullMatrix', 'varname', 'workspace', xreal, ximag)
```

如果 Visual Basic 作为客户程序：

```
void PutFullMatrix([in] BSTR name, [in] BSTR workspace,
    [in] SAFEARRAY(double) xreal, [in] SAFEARRAY(double) ximag);
```

PutFullMatrix 函数将句柄 h 表示的服务器的指定工作空间中的矩阵保存到变量 varname 中。将矩阵的实部和虚部输入到 xreal 和 ximag 输入参数中。workspace 参数可以是 base 或 global。

注意：xreal 和 ximag 参数中指定的矩阵不能是标量、空矩阵或具有 2 维以上。

【例 1】 下面的例子给服务器基本工作空间中的变量 M 赋 5×5 的实数矩阵，然后用 GetFullMatrix 函数读回。实部和虚部（可选）通过单独的 double 型数组传入。

如果 MATLAB 为客户程序，使用下面的命令行：

```
h = actxserver('matlab.application');
h.PutFullMatrix('M', 'base', rand(5), zeros(5))

xreal = h.GetFullMatrix('M', 'base', zeros(5), zeros(5))
xreal =
    0.9501    0.7621    0.6154    0.4057    0.0579
    0.2311    0.4565    0.7919    0.9355    0.3529
```

```
    0.6068    0.0185    0.9218    0.9169    0.8132
    0.4860    0.8214    0.7382    0.4103    0.0099
    0.8913    0.4447    0.1763    0.8936    0.1389
```

如果 Visual Basic 为客户程序，使用类似下面的语句行：

```
Dim MatLab As Object
Dim XReal(5, 5) As Double
Dim XImag(5, 5) As Double
Dim ZReal(5, 5) As Double
Dim ZImag(5, 5) As Double
Dim i, j As Integer

For i = 0 To 4
    For j = 0 To 4
        XReal(i, j)    = Rnd * 6
        XImag(i, j) = 0
    Next j
Next i

Set Matlab = CreateObject("matlab.application")
Call MatLab.PutFullMatrix("M", "base", XReal, XImag)

Call MatLab.OetFullMatrix("M", "base", ZReal, ZImag)
```

【例 2】 上面的例子将矩阵写入基本工作空间，本例将矩阵写入服务器的全局工作空间，然后从客户程序探察服务器的全局工作空间。

MATLAB 作为客户程序时，使用下面的命令行：

```
h.PutFullMatrix('X', 'global', [1 3 5; 2 4 6], [0 0 0; 0 0 0])
h.invoke('Execute', 'whos global')
ans =
  Name        Size                    Bytes  Class
  X            2x3                       96  double array (global complex)
Orand total is 6 elements using 96 bytes
```

Visual Basic 作为客户程序时，使用类似下面的语句行：

```
Dim MatLab As Object
Dim XReal(1 To 3) As Double
Dim XImag(1 To 3) As Double
Dim gblvar As String
Dim i, j As Integer

XReal(1) = 1
XReal(2) = 3
XReal(3) = 5

Set MatLab = CreateObject("matlab.application")
Call MatLab.PutFullMatrix("M", "global", XReal, XImag)
gblvar = MatLab.Execute("whos global")
Print gblvar
```

【例 3】 下面的实例首先建立与 MATLAB 的自动化连接，然后传入一个 3×3 的矩阵，

求该矩阵的逆以后将逆矩阵返回到 VB。

创建新工程 vb_ole_matrix，标准窗体名称为 frmVBOLEMat，在窗体上拖放 1 个框架控件，其中放 9 个文本框，在框架控件右侧放置 2 个标签和 2 个命令按钮，如图 8-4 所示。

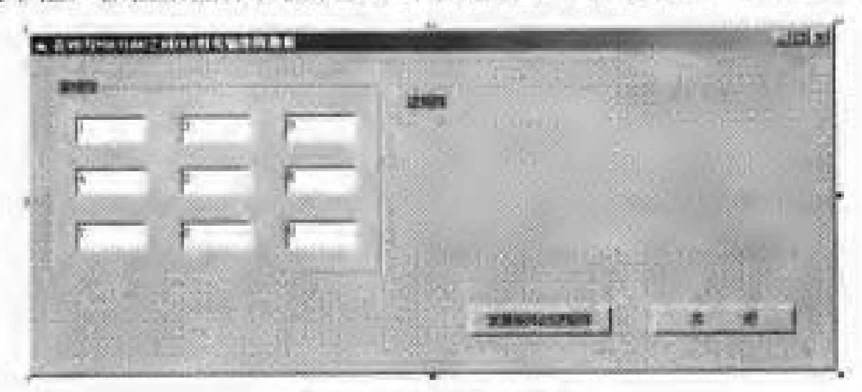


图 8-4 程序的设计界面

窗体和各控件的设置如表 8-2 所示。

表 8-2 窗体和控件的设置

对象类型	名称	属性设置
Form	frmVBOLEMat	Caption="在 VB 与 MATLAB 之间 OLE 时传输矩阵数据"
Frame	Frame1	Caption="原矩阵"
TextBox 数组	txtEle(0)	Text="1"
	txtEle(1)	Text="3"
	txtEle(2)	Text="9"
	txtEle(3)	Text="4"
	txtEle(4)	Text="2"
	txtEle(5)	Text="5"
	txtEle(6)	Text="7"
	txtEle(7)	Text="2"
	txtEle(8)	Text="0"
Label	Label1	Text="逆矩阵"
	Label2	Text=""
CommandButton	cmdCompute	Caption="求原矩阵的逆矩阵"
	cmdExit	Caption="关闭"

在代码窗口键入下面的代码：

```
Private Sub cmdCompute_Click()
  Dim objMATLAB As Object
  Dim dblInReal(1 To 3, 1 To 3) As Double
  Dim dblInImag(1 To 3, 1 To 3) As Double
  Dim dblOutReal(1 To 3, 1 To 3) As Double
  Dim dblOutImag(1 To 3, 1 To 3) As Double
  Dim i As Integer
  Dim j As Integer

  dblInReal(1, 1) = Val(txtEle(0).Text)
  dblInReal(1, 2) = Val(txtEle(1).Text)
```

```
    dblInReal(1, 3) = Val(txtEle(2).Text)
    dblInReal(2, 1) = Val(txtEle(3).Text)
    dblInReal(2, 2) = Val(txtEle(4).Text)
    dblInReal(2, 3) = Val(txtEle(5).Text)
    dblInReal(3, 1) = Val(txtEle(6).Text)
    dblInReal(3, 2) = Val(txtEle(7).Text)
    dblInReal(3, 3) = Val(txtEle(8).Text)

    For i = 1 To 3
        For j = 1 To 3
            dblInImag(i, j) = 0
        Next
    Next

    Set MatLab = CreateObject("matlab.application")
    Call MatLab.PutFullMatrix("InM", "base", dblInReal, dblInImag)
    Call MatLab.Execute("OutM=inv(InM);")
    Call MatLab.GetFullMatrix("OutM", "base", dblOutReal, dblOutImag)

    Dim strOutM As String
    lblOut.Caption = ""
    For i = 1 To 3
        For j = 1 To 3
            strOutM = strOutM & dblOutReal(i, j) & Chr(9)
        Next
        lblOut.Caption = lblOut.Caption & strOutM & Chr(13) & Chr(10)
        strOutM = ""
    Next

    Set MatLab = Nothing
End Sub

Private Sub cmdExit_Click()
    Unload frmVBOLEMat
End Sub
```

运行程序，单击“求原矩阵的逆矩阵”命令按钮，在右侧显示原矩阵的逆矩阵，如图 8-5 所示。



图 8-5　程序运行效果

8.1.3 传递字符串

使用 GetCharArray 函数和 PutCharArray 函数可以在 MATLAB 与 VB 之间传递字符串。

1. GetCharArray 函数

该函数从服务器程序获取字符串，调用格式如下。

如果 MATLAB 用作客户程序：

```
string = h.GetCharArray('varname', 'workspace')
string = GetCharArray(h, 'varname', 'workspace')
string = invoke(h, 'GetCharArray', 'varname', 'workspace')
```

如果 Visual Basic 用作客户程序：

```
void GetCharArray([in] BSTR varname, [in] BSTR workspace,
    [out] BSTR string);
```

GetCharArray 函数获取保存在变量 varname 中的字符串，varname 变量位于句柄 h 表示的服务器的工作空间 workspace 中。获取的值返回到 string 中。workspace 参数可以是 base 或 global。

注意：*如果试图将 GetCharArray 函数的输出显示到客户窗口上，必须指定一个输出变量，即 string。*

2. PutCharArray 函数

使用该函数将字符串保存到服务器中，调用格式如下。

如果 MATLAB 为客户程序：

```
h.PutCharArray('varname', 'workspace', 'string')
PutCharArray(h, 'varname', 'workspace', 'string')
invoke(h,'PutCharArray', 'varname', 'workspace', 'string')
```

如果 Visual Basic 为客户程序：

```
void PutCharArray([in] BSTR name, [in] BSTR workspace,[in] BSTR string);
```

PutCharArray 函数将服务器 h 的工作空间 workspace 中 string 参数内的字符串保存到变量 varname 中。workspace 参数可以是 base 或 global。

注意：*string 参数指定的字符串可以具有任意维数。*

下面的例子用 PutCharArray 函数将字符串指定给服务器基本工作空间的变量 str 中。然后用 GetCharArray 函数读回到客户程序。

MATLAB 用作客户程序时，使用下面的命令行：

```
h = actxserver('matlab.application');
h.PutCharArray('str', 'base', ...
    'He jests at scars that never felt a wound.');

S = h.GetCharArray('str', 'base')
S =
  He jests at scars that never felt a wound.
```

Visual Basic 用作客户程序时，使用类似下面的语句行：

```
Dim Matlab As Object
Dim S As String
```

```
Set Matlab = CreateObject("matlab.application")
Call Matlab.PutCharArray("str", "base","He jests at scars that never felt a wound.")

Call Matlab.GetCharArray("str", "base", S)
```

8.1.4 处理工作空间的数据

使用 GetWokspace 函数和 PutWorkspace 函数可以从工作空间获取数据或者将数据放到工作空间上。

1．GetWorkspaceData 函数

该函数从服务器工作空间获取数据，调用格式如下。

如果 MATLAB 为客户程序：

```
D = h.GetWorkspaceData('varname', 'workspace')
D = GetWorkspaceData(h, 'varname', 'workspace')
D = invoke(h,'GetWokspaceData', 'varname', 'workspace')
```

如果 Visual Basic 作为客户程序：

```
void GetWorkspaceData([in] BSTR varname, [in] BSTR workspace,
    [out] BSTR retdata);
```

GetWorkspaceData 函数获取保存在变量 varname 中的数据，该变量位于句柄 h 表示的服务器的工作空间 workspace 中。返回值返回到输出参数 D 中。workspace 参数可以是 base 或 global。

下面的例子将一个单元数组赋给服务器基本工作空间中的变量 C1，然后用 GetWokspaceData 函数读回。

MATLAB 作为客户程序时，使用下面的命令行：

```
h = actxserver('matlab.application');
h.PutWorkspaceData('C1', 'base', {25.72, 'hello', rand(4)});

C2 = h.GetWorkspaceData('C1', 'base')
C2 =
    [25.7200]    'hello'    [4x4 double]
```

Visual Basic 作为客户程序时，使用类似下面的语句行：

```
Dim Matlab As Object

Set Matlab = CreateObject("matlab.application")
Result = MatLab.Execute("C1 = {25.72, 'hello', rand(4)};")

Call Matlab.GetWorkspaceData("C1", "base", C2)
```

2．PutWorkspaceData 函数

使用该函数将数据保存到服务器工作空间中，调用格式如下。

MATLAB 用作客户程序时：

```
h.PutWorkspaceData('varname', 'workspace', data)
PutWorkspaceData(h, 'varname', 'workspace', data)
```

```
invoke(h, 'PutWorkspaceData', 'varname', 'workspace', data)
```

Visual Basic 用作客户程序时：

```
void PutWorkspaceData([in] BSTR name, [in] BSTR workspace,[in] BSTR data);
```

PutWorkspaceData 函数将句柄 h 表示的服务器的工作空间 workspace 中的数据保存到变量 varname 中。workspace 参数可以是 base 或 global。

下面的例子在客户程序中创建一个数组，并将它赋给服务器基本工作空间中的变量 A。

MATLAB 用作客户程序时，使用下面的命令行：

```
h = actxserver('matlab.application');

for i = 1:6
    data(i)  = i .* 15;
end

h.PutWorkspaceData('A', 'base', data);
```

Visual Basic 用作客户程序时，使用类似下面的语句行：

```
Dim Matlab As Object
Dim data(7) As Double

Set Matlab = CreateObject("matlab.application")
For i = 0 To 6
    data(i)  = i * 15;
Next i

Call Matlab.PutWorkspaceData("A", "base", data)
```

8.1.5 传递和处理 MATLAB 函数

使用 Feval 函数在服务器中处理 MATLAB 函数，该函数的调用格式如下。

如果 MATLAB 用作客户程序：

```
result = h.Feval('functionuame', numout, arg1, arg2, ...)
result = Feval(h, 'functionname', numout, arg1, arg2, ...)
result = invoke(h,'Feval', 'functionname', numout, arg1, arg2, ...)
```

如果 Visual Basic 用作客户程序：

```
void Feval([in] BSTR functionname, [in] long numout,
    [out] VARIANT* result, [in] VARIANT arg1, arg2, ...)
```

Feval 函数在句柄 h 表示的自动化服务器中执行字符串 functionname 指定的 MATLAB 函数。1×1 的 double 型数组 numout 为函数返回的输出参数个数。服务器将函数的输出返回到单元数组 result 中。

最多可以指定 32 个输入参数，这些参数放在 numout 参数的后面。传递参数有 4 种方式，如表 8-3 所示。

下面举例说明 Feval 函数的使用。

（1）MATLAB 用作客户程序时的参数传递

① 在 Feval 函数中通过传递 strcat 函数名来聚合两个单词。

```
a = h.Feval('strcat', 1, 'hello ', 'world')
a =
    'hello world'
```

表 8-3　参数传递机制

传递机制	描　述
传递值本身	传递任何数值或字符串值，在 Feval 参数列表中指定值： a = h.Feval('sin', 1, -pi:0.01:pi);

传递客户变量	传递一个参数，该参数已经在客户程序中赋给变量，单独指定变量名：
	x = -pi:0.01:pi;
	a = h.Feval('sin', 1, x);

引用服务器变量	引用服务器中定义的变量，指定后面带等号的变量名：
	h.PutWorkspaceData('x', 'base', -pi:0.01:pi);
	a = h.Feval('sin', 1, 'x=');

引用并覆盖服务器变量	引用服务器中定义的变量并在服务器中覆盖该变量，指定变量名、等号和要赋给变量的新值：
	h.PutWorkspaceData('x', 'base', -pi:0.01:pi);
	a = h.Feval('sin', 1, 'x=-5:0.01:5');

② 进行相同的聚合，传递一个字符串和一个局部变量 clistr，clistr 包含第 2 个字符串。

```
clistr = 'world';
a = h.Feval('strcat', 1, 'hello ', clistr)
a =
    'hello world'
```

③ 本例中 srvstr 变量在服务器程序，而不是客户程序中定义。在变量名后面加一个等号，告诉 MATLAB 这是一个服务器变量，MATLAB 就不会将它作为局部变量处理。

```
% 在服务器中定义变量 srvstr
h.PutCharArray('srvstr', 'base', 'world');

% 用'name='语法格式传递服务器变量的名称
a = h.Feval('strcat', 1, 'hello ', 'srvstr=')
a =
    'hello world'
```

④ 本例与上例类似，但要给服务器变量赋新值，然后用该新值进行聚合。

```
% 在服务器上定义变量 srvstr
h.PutCharArray('srvstr', 'base', 'world');

% 给 srvstr 赋一个新字符串并进行聚合
a = h.Feval('strcat', 1, 'hello ', 'srvstr=everyone')
a =
    'hello everyone'

% 修改 srvstr 变量，它现在有新值
s = h.GetCharArray('srvstr', 'base')
s =
    'everyone'
```

（2）Visual Basic 用作客户程序时

下面将 Visual Basic 作为客户程序，实现与上面例子相同的功能。这些例子返回与上面的

相同的字符串。

① 传递两个字符串：

```
Dim Matlab As Object
Set Matlab = CreateObject("matlab.application")
Call Matlab.Feval("strcat", 1, out, "hello ", "world")
```

② 局部定义 clistr 并传递该变量：

```
Dim Matlab As Object
Dim clistr As String
Set Matlab = CreateObject("matlab.application")

clistr = "world"
Call Matlab.Feval("strcat", 1, out, "hello ", "clistr")
```

③ 传递在服务器上定义的变量的名称。

```
Dim Matlab As Object
Set Matlab = CreateObject("matlab.application")
Call Matlab.PutCharArray("srvstr", "base", "world")
Call Matlab.Feval("strcat", 1, out, "hello ", "srvstr=")
```

④ 传递在服务器上定义的变量的名称，还给该变量提供一个新值。

```
Dim Matlab As Object
Set Matlab = CreateObject("matlab.application")
Call Matlab.PutCharArray("srvstr", "base", "world")
Call Matlab.Feval("strcat", 1, out, "hello ", "srvstr=","everyone")
Call Matlab.GetCharArray("srvstr", "base", s)
```

客户程序为 MATLAB 或 VB 时，Feval 函数的返回值有如下定义：

MATLAB 用作客户程序时：

Feval 函数将处理函数的返回数据放在一个单元数组中。每个返回值在单元数组中占一行。可以用 Feval 函数的第 2 个输入参数控制返回值的个数，如本例所示。本例要求 Feval 函数从 fileparts 函数返回 3 个输出参数。需要注意的是，要求返回的参数个数不能比函数返回值的最大个数多。

```
a = h.Feval('fileparts', 3, 'd:\work\ConsoleApp.cpp')
a =
    'd:\work'
    'ConsoleApp'
    '.cpp'
```

Visual Basic 用作客户程序时：

下面是相同的例子，但用 Visual Basic 编码。

```
Dim Matlab As Object
Set Matlab = CreateObject("matlab.application")
Call Matlab.Feval("fileparts", 4, out, "d:\work\ConsoleApp.cpp")
```

8.1.6 其他操作

使用 MaximizeCommandWindow 等函数，可以实现服务器窗口显示、最小化和终止运行等功能。

1．MaximizeCommandWindow 函数

该函数在 Windows 桌面上显示服务器窗口，语法格式如下。

如果 MATLAB 为客户程序：

```
h.MaximizeCommandWindow
MaximizeCommandWindow(h)
invoke(h,'MaximizeCommandWindow')
```

如果 Visual Basic 作为客户程序：

```
void MaximizeCommandWindow;
```

MaximizeCommandWindow 函数显示句柄为 h 的服务器程序的窗口，并使它成为当前激活的窗口。如果服务器窗口并不处于最小化状态，则 MaximizeCommandWindow 函数不起作用。

下面的例子创建一个 COM 服务器，使它的窗口最小化，然后最大化窗口并使它成为当前激活的窗口。

MATLAB 为客户程序时，使用下面的命令行：

```
h = actxserver('matlab.application');
h.MinimizeCommandWindow;
% 现在，将服务器窗口返回到前一状态
% 并使之成为当前激活的窗口
h.MaximizeCommandWindow;
```

Visual Basic 作为客户程序时，使用类似下面的语句行：

```
Dim Matlab As Object

Set Matlab = CreateObject("matlab.application")
Call Matlab.MinimizeCommandWindow

rem 现在，将服务器窗口返回到前一状态
rem  并使之成为当前激活的窗口

Call Matlab.MaximizeCommandWindow
```

2．MinimizeCommandWindow 函数

该函数使服务器窗口最小化，调用格式与 MaximizeCommandWindow 函数的相同。

3．Quit 函数

该函数终止 MATLAB 服务器的运行。调用格式如下。

如果 MATLAB 为客户程序：

```
h.Quit
Quit(h)
invoke(h,'Quit')
```

如果 Visual Basic 为客户程序：

```
void Quit
```

8.2 基于 ActiveX 的接口实现

ActiveX 技术是由 OLE 技术发展而来的，它使应用程序之间的通信超出了本地机的范围。相关技术包括 ActiveX 控件、ActiveX DLL、ActiveX EXE 和 ActiveX 文档等。MATLAB 对前面 3 种方式提供了较好的支持。

8.2.1 使用 ActiveX 控件

在 MATLAB 中，使用 actxcontrol 函数可以创建 ActiveX 控件的实例，并显示到图形窗口中，从而使 MATLAB 界面设计增加了无限的可能性。第 15 章会进一步讨论这方面的问题。第 6 章介绍了 actxcontrol 函数的调用格式。

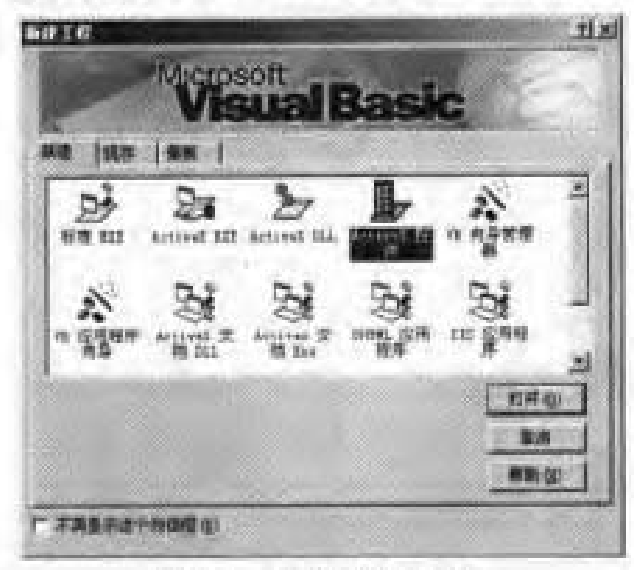


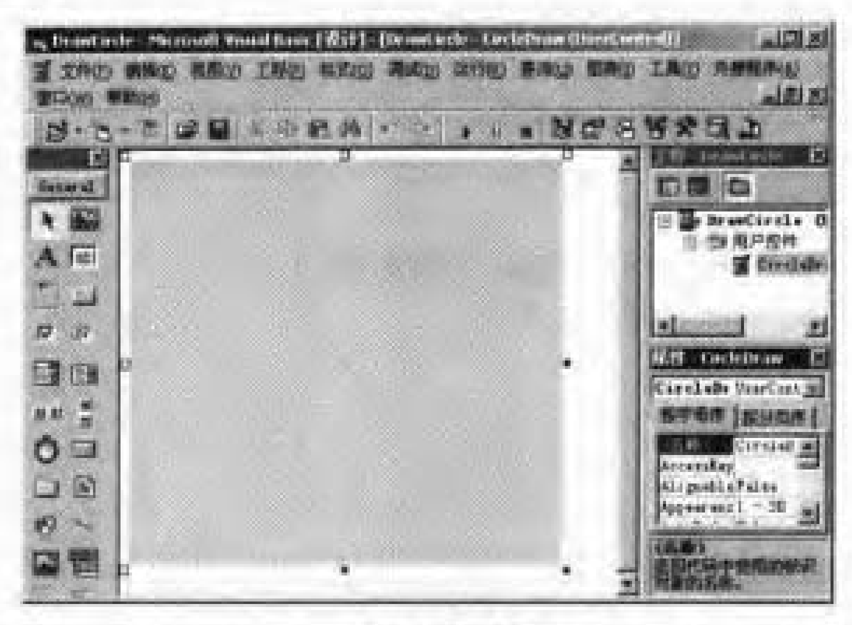
图 8-6 “新建工程”对话框

下面的例子首先用 VB 创建一个画圆的控件，然后 MATLAB 作为客户端调用该控件。

1. 用 VB 创建控件

下面在 Visual Basic 中创建一个画圆的 ActiveX 控件。如图 8-6 所示，当进入 Visual Basic 环境时，在“新建工程”对话框中选择“ActiveX 控件”图标，然后单击“打开”按钮，确定创建一个 ActiveX 控件。

然后在控件中添加一个蓝色图像框 picDraw，如图 8-7 所示。注意将 picDraw 的 AutoRedraw 属性设置为 True。这个图像框将作为我们画圆时的画布。把该控件保存为 CircleDraw.ctl。


图 8-7 设计控件外观

下一步添加控件的属性和方法。因为是画圆，所以要确定圆的圆心坐标，以确定圆的位置。另外还要确定圆的半径，以确定圆的大小。所以，给 CircleDraw 控件添加三个属性，即 CenterX、CenterY 和 CircleR，其中，CenterX 和 CenterY 分别是圆心的横坐标和纵坐标，CircleR 为圆的半径。定义圆的位置和大小以后，还要定义一个 Draw 方法，实现绘图。

在代码窗口中添加下面的代码：

```
Option Explicit

Dim m_CenterX As Double
Dim m_CenterY As Double
Dim m_CircleR As Double

Public Property Let CenterX(NewData As Double)
  m_CenterX = NewData
End Property

Public Property Get CenterX() As Double
  CenterX = m_CenterX
End Property

Public Property Let CenterY(NewData As Double)
  m_CenterY = NewData
End Property

Public Property Get CenterY() As Double
  CenterY = m_CenterY
End Property

Public Property Let CircleR(NewData As Double)
  m_CircleR = NewData
End Property

Public Property Get CircleR() As Double
  CircleR = m_CircleR
End Property

Public Sub Draw()
  picDraw.Circle (m_CenterX, m_CenterY), m_CircleR
End Sub
```

最后保存控件到文件，并单击“文件”菜单中的“生成 CircleDraw.ocx...”选项，生成 ActiveX 控件 CircleDraw.ocx。生成控件的过程中，系统会自动注册该控件，这样，在其他工程中就可以引用它。

2．测试控件

控件制作完毕以后，为了检查它是否能正常工作，一般要对它进行测试。下面我们就测试一下刚刚建立的 CircleDraw.ocx 控件。

首先，在 Visual Basic 中建立一个标准工程 ctlTest.vbp。然后从“工程”菜单中单击“部件...”选项，打开“部件”对话框，如图 8-8 所示。利用该对话框，可以把可用的部件导入到

当前工程中。我们需要的 CircleDraw.ocx 控件在“控件”选项卡中没有出现。单击“浏览...”按钮，从该控件的存放目录中找到它并确定打开，则 CircleDraw.ocx 控件的服务名 DrawCircle 显示在“控件”选项卡中，并且被选中，如图 8-8 所示。

在“部件”对话框中单击“确定”按钮，CircleDraw.ocx 控件被导入到当前工程中来了，并在窗体左侧的工具栏中显示它的图标。试着像使用基本控件那样使用它。单击它以后用鼠标在设计窗体（Form1）中进行拖拉操作，显示该控件的界面，即一张蓝色的画布。现在，再添加一个命令按钮 cmdTest，将它放在绘图按钮正下方，标题为“测　试”。效果如 8-9 所示。



图 8-8　“部件”对话框



图 8-9　测试程序窗体设计

下一步添加功能代码，在 cmdTest 命令按钮的单击事件代码框架中添加下面的代码，利用画图控件画圆。

```
Private Sub cmdTest_Click()
  With CircleDraw1
    .CenterX = 1000
    .CenterY = 1000
    .CircleR = 500
    .Draw
    .CenterX = 2500
    .CenterY = 1500
    .CircleR = 1000
    .Draw
  End With
End Sub
```

运行程序，在弹出的程序窗体中单击“测试”按钮，在兰色画布上画一个圆，如图 8-10 所示。这样的结果，说明我们自己做的控件 CircleDraw.ocx 能够正常工作。

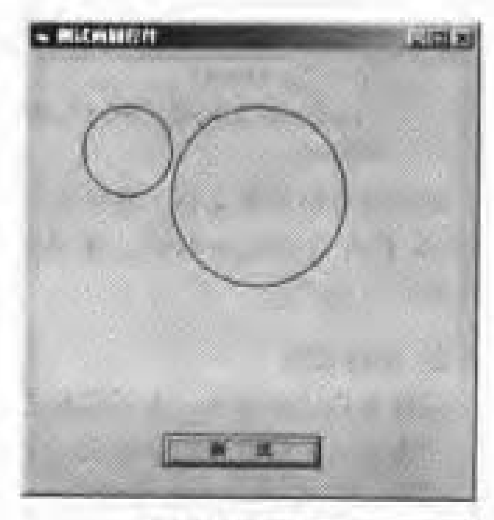


图 8-10　测试结果

3．在 MATLAB 中使用 CircleDraw 控件

现在在 MATLAB 中使用 CircleDraw 控件，包括导入组件、信息获取、属性修改和重绘图形等各项工作。

用 actxcontrol 函数在 MATLAB 中创建 ActiveX

自动化控件。在命令窗口键入

```
h=actxcontrol('drawcircle.circledraw',[0,0,300,300])
h =
     COM.drawcircle.circledraw
```

如图 8-11 所示，CircleDraw 控件被导入到图形窗口中。

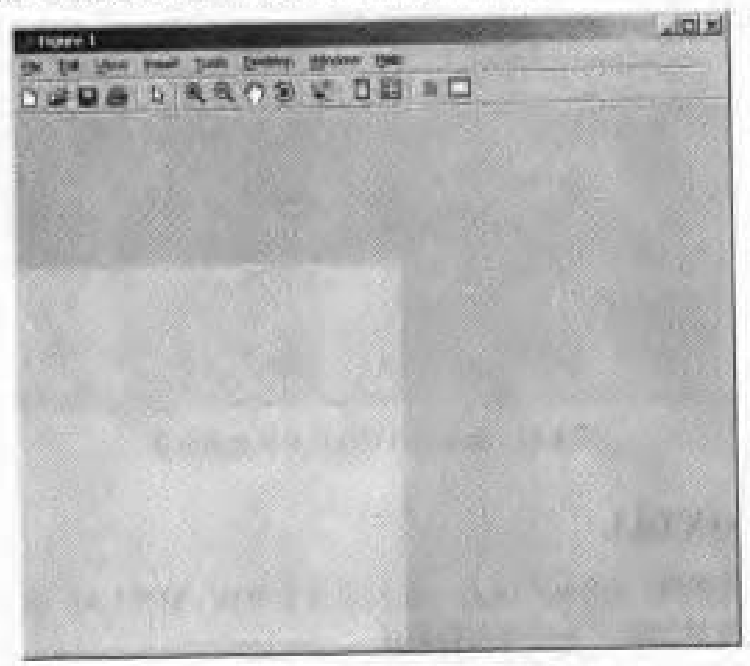


图 8-11 在 MATLAB 图形窗口中载入控件

用 get 函数和 invoke 函数分别获取控件的属性和方法。

```
h.get
     CenterX: 0
     CenterY: 0
     CircleR: 0

h.invoke
     Draw = void Draw(handle)
```

从上面的属性值中可以看出，默认时，圆心坐标和半径等属性的值都是 0，所以图 8-11 中没有看到圆。

下面用 MATLAB 的 set 函数修改控件的属性，并用 Draw 方法进行重绘。这里画了一大一小两个圆。

```
h.CenterX=1000;
h.CenterY=1000;
h.CircleR=500;
h.Draw
h.CenterX=2500;
h.CenterY=1500;
h.CircleR=1000;
h.Draw
```

结果如图 8-12 所示。

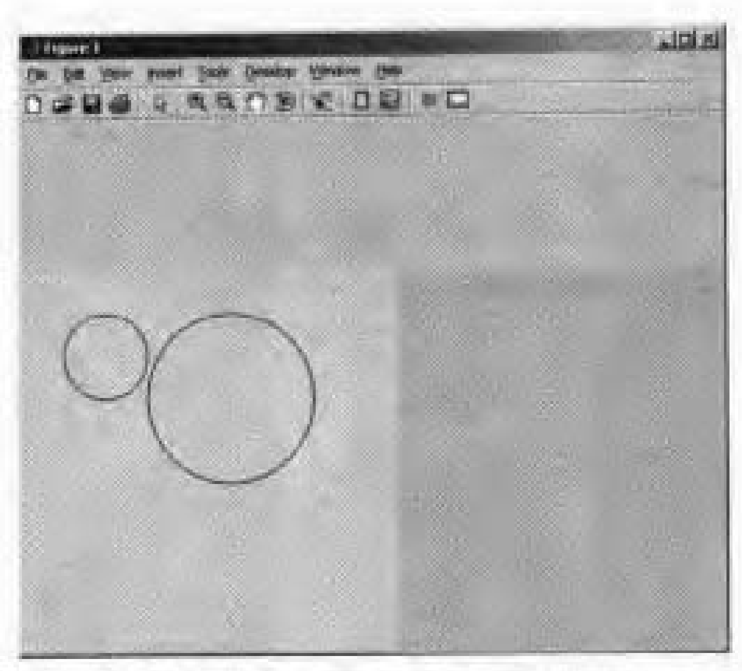


图 8-12 修改控件属性以后的绘图效果

8.2.2 使用 ActiveX DLL

与 ActiveX 控件不同，ActiveX DLL 一般没有图形界面。MATLAB 中用 actxserver 函数创建 ActiveX DLL 对象的实例，为进程内的集成。

下面创建的 DLL 库是一个三维图形数学库的一部分，库中有 2 个类，即 Vector3D 类和 Matrix3D 类，分别描述矢量和矩阵计算。

1. 创建 ActiveX DLL

打开 VB 程序时，在"新建工程"模板中选择"ActiveX DLL"图标，单击"打开"按钮，可以创建一个 ActiveX DLL 工程，如图 8-13 所示。

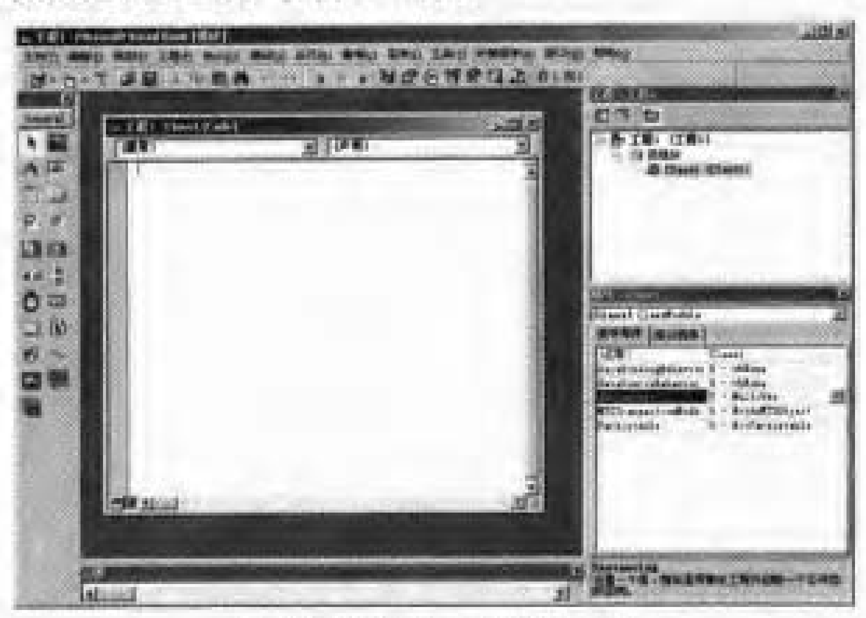


图 8-13 创建 ActiveX DLL 工程

与 ActiveX 控件不同的是，ActiveX DLL 没有界面，只有类代码。默认时，VB 会自动在

工程中添加一个 Class1 类，该类有 DataBindingBehavior 等 5 个属性。新建一个 ActiveX DLL 工程，名称为 Math3D。

将自动生成的 Class1 类的名称改为 Vector3D，在代码窗口添加实现代码，如下所示。x、y、z 属性分别表示三维矢量在 x、y、z 三个方向上的值。

```
Option Explicit

Private mdblX As Double
Private mdblY As Double
Private mdblZ As Double

Public Property Let z(ByVal vData As Double)
   mdblZ = vData
End Property

Public Property Get z() As Double
   z = mdblZ
End Property

Public Property Let y(ByVal vData As Double)
   mdblY = vData
End Property

Public Property Get y() As Double
   y = mdblY
End Property

Public Property Let x(ByVal vData As Double)
   mdblX = vData
End Property

Public Property Get x() As Double
   x = mdblX
End Property
```

Normalize 过程对矢量进行归一化，需要调用 GetLength 函数获取矢量的长度。

```
'归一化
Public Sub Normalize()
   Dim dblLen As Double
   dblLen = GetLength
   mdblX = mdblX / dblLen
   mdblY = mdblY / dblLen
   mdblZ = mdblZ / dblLen
End Sub
```

GetNormal 函数返回将当前矢量归一化后的矢量。

```
'获取归一化矢量
Public Function GetNormal() As Vector3D
   Dim dblLen As Double
   dblLen = GetLength
   GetNormal.x = mdblX / dblLen
```

```
        GetNormal.y = mdblY / dblLen
        GetNormal.z = mdblZ / dblLen
    End Function
```

GetLengthZX,GetLengthYZ,和 GetLengthXY 函数分别返回矢量在 ZX,YZ 和 XY 平面上的投影长度，GetLength 函数返回矢量的长度。

```
    '获取矢量在 ZX 面上的投影长度
    Public Function GetLengthZX() As Double
        GetLengthZX = Sqr(mdblZ * mdblZ + mdblX * mdblX)
    End Function

    '获取矢量在 YZ 面上的投影长度
    Public Function GetLengthYZ() As Double
        GetLengthYZ = Sqr(mdblY * mdblY + mdblZ * mdblZ)
    End Function

    '获取矢量在 XY 面上的投影长度
    Public Function GetLengthXY() As Double
        GetLengthXY = Sqr(mdblX * mdblX + mdblY * mdblY)
    End Function

    '获取矢量的长度
    Public Function GetLength() As Double
        GetLength = Sqr(mdblX * mdblX + mdblY * mdblY + mdblZ * mdblZ)
    End Function
```

下面几个函数实现矢量运算，包括

- Devide 函数返回矢量除以常数以后的结果；
- MultDot 返回矢量点乘的结果；
- MultCross 函数返回矢量叉乘的结果；
- MultCon 函数返回矢量乘以常数的结果；
- Subtract 函数返回矢量相减的结果；
- Add 函数返回矢量相加的结果。

```
    '矢量除以常数
    Public Function Devide(dblC As Double) As Vector3D
        Dim vctD As Vector3D
        Set vctD = New Vector3D

        vctD.x = mdblX / dblC
        vctD.y = mdblY / dblC
        vctD.z = mdblZ / dblC

        Set Devide = vctD
    End Function

    '矢量点乘
    Public Function MultDot(vctV As Vector3D) As Double
        MultDot = mdblX * vctV.x + mdblY * vctV.y + mdblZ * vctV.z
    End Function
```

```
'矢量叉乘
Public Function MultCross(vctV As Vector3D) As Vector3D
  Dim vctM As Vector3D
  Set vctM = New Vector3D

  vctM.x = mdblY * vctV.z - vctV.y * mdblZ
  vctM.y = mdblZ * vctV.x - vctV.z * mdblX
  vctM.z = mdblX * vctV.y - vctV.x * mdblY

  Set MultCross = vctM
End Function

'矢量乘以常数
Public Function MultCon(dblC As Double) As Vector3D
  Dim vctM As Vector3D
  Set vctM = New Vector3D

  vctM.x = mdblX * dblC
  vctM.y = mdblY * dblC
  vctM.z = mdblZ * dblC

  Set MultCon = vctM
End Function

'矢量相减
Public Function Subtract(vctV As Vector3D) As Vector3D
  Dim vctS As Vector3D
  Set vctS = New Vector3D

  vctS.x = mdblX - vctV.x
  vctS.y = mdblY - vctV.y
  vctS.z = mdblZ - vctV.z

  Set Subtract = vctS
End Function

'矢量相加
Public Function Add(vctV As Vector3D) As Vector3D
  Dim vctA As Vector3D
  Set vctA = New Vector3D

  vctA.x = mdblX + vctV.x
  vctA.y = mdblY + vctV.y
  vctA.z = mdblZ + vctV.z

  Set Add = vctA
End Function
```

在“工程”菜单中单击“添加类模块”选项，打开“添加类模板”对话框，单击“确定”

按钮，添加一个 Class1 类。将类名改为 Matrix3D，在代码窗口输入实现代码。

Matrix3D 类有一个 M 属性，它是一个二维属性，需要给出横向和纵向的索引值。

```
Option Explicit

Private mdblM(0 To 3, 0 To 3) As Double

Public Property Let M(ByVal intI As Integer, ByVal intJ As Integer, ByVal dblData As Double)
      mdblM(intI, intJ) = dblData
End Property

Public Property Get M(ByVal intI As Integer, ByVal intJ As Integer) As Double
      M = mdblM(intI, intJ)
End Property
```

GetValue 函数计算矩阵行列式的值并返回。

```
'计算行列式的值
Public Function GetValue() As Double
   GetValue = mdblM(0, 0) * mdblM(1, 1) * mdblM(2, 2) + _
                  mdblM(0, 1) * mdblM(1, 2) * mdblM(2, 0) + _
                  mdblM(0, 2) * mdblM(1, 0) * mdblM(2, 1) + _
                  mdblM(0, 2) * mdblM(1, 1) * mdblM(2, 0) + _
                  mdblM(0, 1) * mdblM(1, 0) * mdblM(2, 2) + _
                  mdblM(0, 0) * mdblM(1, 2) * mdblM(2, 1)
End Function
```

IdenticalMatrix 过程生成单位矩阵。

```
'单位矩阵
Public Sub IdenticalMatirx()
   Dim intI As Integer
   Dim intJ As Integer
   For intI = 0 To 3
      For intJ = 0 To 3
         If intI = intJ Then
            mdblM(intI, intJ) = 1#
         Else
            mdblM(intI, intJ) = 0#
         End If
      Next
   Next
End Sub
```

Multiply 函数计算矩阵相乘的结果并返回。

```
'矩阵相乘
Public Function Multiply(mtxM As Matrix3D) As Matrix3D
   Dim mtxMult As Matrix3D
   Set mtxMult = New Matrix3D

   Dim intI As Integer
   Dim intJ As Integer
   For intI = 0 To 3
```

```
            For intJ = 0 To 3
                mtxMult.M(intI, intJ) = mdblM(intI, 0) * mtxM.M(0, intJ) + _
                                        mdblM(intI, 1) * mtxM.M(1, intJ) + _
                                        mdblM(intI, 2) * mtxM.M(2, intJ) + _
                                        mdblM(intI, 3) * mtxM.M(3, intJ)
            Next
        Next

        Set Multiply = mtxMult
    End Function
```

TransT, ScaleT, RotateT 和 MirrorT 函数分别计算平移变换、比例变换、旋转变换和镜像变换矩阵。

```
'计算平移变换矩阵
Public Function TransT(vctV As Vector3D) As Matrix3D
    '添加代码
End Function

'计算比例变换矩阵
Public Function ScaleT(dblS As Double) As Matrix3D
    '添加代码
End Function

'计算旋转变换矩阵
Public Function RotateT(vctV As Vector3D, dblD As Double) As Matrix3D
    '添加代码
End Function

'计算镜像变换矩阵
Public Function MirrorT(vctV As Vector3D) As Matrix3D
    '添加代码
End Function
```

类的添加完成以后，在工程的“文件”菜单中单击“生成 Math3D.dll...”选项，生成动态链接库 Math3D.dll。

2. 测试 DLL

生成 ActiveX DLL 后，需要对该库进行测试。可以直接在工程中添加工程进行测试，也可以另外建立工程进行测试。下面另外建立一个新工程测试 Math3D 库。

新建标准 EXE 工程 DLLTest，窗体名称为 frmDLLTest。在“工程”菜单中单击“引用...”选项，打开“引用”对话框。在该对话框中利用“浏览...”按钮找到前面生成的 Math3D 库并选择它，如图 8-14 所示。



图 8-14 “引用”对话框

在窗体模块中输入下面的代码，对 Math3D 库中的类进行测试。

```
Option Explicit

Private Sub Form_Click()
    Dim vctFirst As Vector3D
    Dim vctSecond As Vector3D
    Dim mtxFirst As Matrix3D
    Dim mtxSecond As Matrix3D

    Set vctFirst = New Vector3D
    Set vctSecond = New Vector3D
    Set mtxFirst = New Matrix3D
    Set mtxSecond = New Matrix3D

    '第一个矢量
    vctFirst.x = 10#
    vctFirst.y = 78.5
    vctFirst.z = 102.9

    '第二个矢量
    vctSecond.x = 109.2
    vctSecond.y = 82.5
    vctSecond.z = 180.8

    '在窗体上输出第一个矢量和第二个矢量
    Dim strFirst As String
    Dim strSecond As String
    strFirst = "第一个矢量： " & Str(vctFirst.x) & " " & Str(vctFirst.y) & "    " & Str(vctFirst.z)
    strSecond = "第二个矢量： " & Str(vctSecond.x) & " " & Str(vctSecond.y) & _
" " & Str(vctSecond.z)
    Print
    Print strFirst
    Print strSecond
    Print

    '第一个矢量的长度
    Dim strLen As String
    strLen = "第一个矢量的长度： " & vctFirst.GetLength
    Print strLen

    '矢量运算
    Dim vctAdd As Vector3D
    Dim vctSub As Vector3D
    Dim dblMulDot As Double
    Dim vctMulCro As Vector3D
    Dim vctDev As Vector3D
    Set vctAdd = vctFirst.Add(vctSecond)
    Set vctSub = vctFirst.Subtract(vctSecond)
    dblMulDot = vctFirst.MultDot(vctSecond)
    Set vctMulCro = vctFirst.MultCross(vctSecond)
    Set vctDev = vctFirst.Devide(2)
```

```
'输出运算结果
Dim strAdd As String
Dim strSub As String
Dim strMulDot As String
Dim strMulCro As String
Dim strDev As String

strAdd = "矢量的和：" & Str(vctAdd.x) & " " & Str(vctAdd.y) & " " & Str(vctAdd.z)
strSub = "矢量的差：" & Str(vctSub.x) & " " & Str(vctSub.y) & " " & Str(vctSub.z)
strMulDot = "矢量点乘：" & Str(dblMulDot)
strMulCro = "矢量叉乘：" & Str(vctMulCro.x) & " " & Str(vctMulCro.y) & _
" " & Str(vctMulCro.z)
strDev = "第一个矢量除以 2：" & Str(vctDev.x) & " " & Str(vctDev.y) & " " & Str(vctDev.z)

Print strAdd
Print strSub
Print strMulDot
Print strMulCro
Print strDev

'第一个矩阵和第二个矩阵
Dim strFirstM(0 To 3) As String
Dim strSecondM(0 To 3) As String
Dim intI As Integer
Dim intJ As Integer
For intI = 0 To 3
  For intJ = 0 To 3
    mtxFirst.M(intI, intJ) = intI + intJ
    mtxSecond.M(intI, intJ) = intJ - intI
    strFirstM(intI) = strFirstM(intI) & Str(mtxFirst.M(intI, intJ)) & "    "
    strSecondM(intI) = strSecondM(intI) & Str(mtxSecond.M(intI, intJ)) & "    "
  Next
Next

'输出两个矩阵
Print
Print "第一个矩阵："
For intI = 0 To 3
  Print strFirstM(intI)
Next

Print
Print "第二个矩阵："
For intI = 0 To 3
  Print strSecondM(intI)
Next

'计算两个矩阵的乘积
Dim mtxMul As Matrix3D
```

```
    Set mtxMul = mtxFirst.Multiply(mtxSecond)

    '输出乘积
    Print
    Print "两个矩阵的乘积："
    Dim strMulM As String
    For intI = 0 To 3
      For intJ = 0 To 3
        strMulM = strMulM & Str(mtxMul.M(intI, intJ)) & "   "
      Next
      Print strMulM
      strMulM = ""
    Next
End Sub
```

运行程序，单击窗体，在窗体上输出矢量、矩阵数据及其计算结果，如图 8-15 所示。



图 8-15　测试数学库

3. 在 MATLAB 中使用 ActiveX DLL

创建并测试 ActiveX DLL 以后，可以在 MATLAB 中用 actxserver 函数创建该 DLL 中类的实例。例如，下面创建 Math3D 库中 Vector3D 类的实例：

```
h1=actxserver('Math3D.Vector3D')
h1 =
    COM.Math3D_Vector3D
```

默认时，该对象的所有属性值都为 0。设置各属性值，用 get 函数可以显示当前的属性值。

```
h1.x=10;h1.y=20;h1.z=8;
h1.get
    z: 8
    y: 20
    x: 10
```

用 invoke 函数可以显示类的方法及其调用格式。

```
h1.invoke
Add = [handle, handle] Add(handle, handle)
Devide = [handle, double] Devide(handle, double)
GetLength = double GetLength(handle)
GetLengthXY = double GetLengthXY(handle)
GetLengthYZ = double GetLengthYZ(handle)
GetLengthZX = double GetLengthZX(handle)
GetNormal = handle GetNormal(handle)
MultCon = [handle, double] MultCon(handle, double)
MultCross = [handle, handle] MultCross(handle, handle)
MultDot = [double, handle] MultDot(handle, handle)
Normalize = void Normalize(handle)
Subtract = [handle, handle] Subtract(handle, handle)
```

下面调用 GetLength 方法求矢量的长度。

```
h1.invoke('GetLength')
ans =
   23.7487
```

actxserver 函数支持一个库中有多个类的情况，例如下面创建 Matrix3D 类的实例。

```
h2=actxserver('Math3D.Matrix3D')
h2 =
    COM.Math3D_Matrix3D
```

用 invoke 函数显示该对象的所有方法。

```
h2.invoke
GetValue = double GetValue(handle)
IdenticalMatirx = void IdenticalMatirx(handle)
M = double M(handle, int16, int16)
MirrorT = [handle, handle] MirrorT(handle, handle)
Multiply = [handle, handle] Multiply(handle, handle)
RotateT = [handle, handle, double] RotateT(handle, handle, double)
ScaleT = [handle, double] ScaleT(handle, double)
TransT = [handle, handle] TransT(handle, handle)
```

8.2.3 使用 ActiveX EXE

ActiveX EXE 是可执行的，常常用于实现 DCOM。使用 actxserver 函数的第 2 种语法格式可以创建 ActiveX EXE 的实例。其基本操作步骤与前面两种组件的相似，不再详细进行介绍。需要注意的是，使用 ActiveX EXE 时，使用的是进程外的集成方式。

8.3 基于 COM 组件的接口实现

实际上，上一节介绍的 ActiveX 技术就属于 COM 组件技术的范畴。该节主要讨论了在 MATLAB 中调用 VB 组件的问题，本节则讨论在 VB 中调用 MATLAB 组件的技术。这两节内容介绍的是 MATLAB 接口方面的最新技术，建议用心掌握。

8.3.1 使用 COM 生成器

从 6.5 版本开始，MATLAB 提供了 COM 生成器。使用该生成器，可以将大部分 MATLAB 函数和自定义函数打包成组件，然后集成到支持该技术的应用程序中去。使用该技术，可以在很大程度上脱离 MATLAB 环境。

第 13 章详细介绍 COM 生成器的用法，请参阅。

8.3.2 关于 MatrixVB

COM 生成器出现以前，Mathworks 提供 MatrixVB 组件实现组件式集成。该组件具备数值计算、绘图等多方面的能力，在一定的历史阶段发挥了它的作用。但 COM 生成器出现以后，Mathworks 公司不再将 MatrixVB 作为单独的产品提供，因为 COM 生成器已经可以取代该组件的功能。

相对于 MatrixVB，使用 COM 生成器具有以下优点：

- 一般来讲，用 COM 生成器生成的组件会比 MatrixVB 小得多。如果希望用 MATLAB 帮助绘一下图，或者进行某些单一的矩阵运算，在以前需要载入整个 MatrixVB，其中包括很多根本用不着的函数。现在使用 COM 生成器就可以“量身定做”了，组件本身不会有多余的东西。

- 由于 COM 生成器是一个生产工具，而 MatrixVB 是一个产品，所以使用 COM 生成器具有大得多的灵活性。

第 9 章　MATLAB 与 Visual C++接口

MATLAB 7 版本自带了 C 语言编译器(Compiler)，可以将 M 文件转换为 C-MEX 或 C/C++程序。根据 MATLAB 是否运行，一般将 MATLAB 与 VC 接口分为两大类：需 MATLAB 在后台运行的混合编程接口和可以脱离 MATLAB 环境运行的独立应用程序接口。本章主要介绍这两种接口的实现。

9.1　MATLAB 与 VC 混合编程接口

MATLAB 与 VC 混合编程接口，主要有如第 7 章所述的 MEX 文件、engine 应用程序和 MAT 文件三种。由于 VC 的内存管理及类库函数比 C 更加灵活方便，因此，对这三种接口方式在 VC 中的实现，主要基于内存管理和类库函数应用方面阐述。

9.1.1　VC 与 MEX 文件示例一

本节是基于第 7 章的 MEX 文件示例“H1.c”经过局部修改而得，文件名改为“H1.cpp”。表明，该 MEX 文件是 MATLAB 与 C++的接口。与一般的 C 接口程序相类似，“H1.cpp”文件也包括三个部分：

（1）头文件：与一般的 C 接口程序一样，必须包含“mex.h”头文件；

（2）用户 Cpp 程序：内存分配函数为 new，释放函数为 delete，用法与普通 Cpp 程序一样；

（3）接口函数 mexFunction：与一般的 C 接口程序一样，也是基于 mxArray 对象进行数据传送。

“H1.cpp”程序清单如下：

```
//////////////////MEX 头文件/////////////////////////////////////////
#include "mex.h"
/////////////////用户 Cpp 程序 /////////////////////////////////////////
double OutPut(int n,double *rP)
{ int i;   double *sum,sum0;
FILE *fp1;
sum=new double[1];/////////////////内存在管理机制为 Cpp 机制
if((fp1=fopen("OutPut1.txt","w"))==NULL){printf(" Can't open OutPutfile\n");exit(0);}
*sum=0.0;
fprintf(fp1,"Input  %d Data is:",n);
for(i=0;i<n;i++){*sum=*sum+rP[i];fprintf(fp1,"  %7.2lf ",rP[i]);}
fprintf(fp1,"\nsum=  %7.2lf \n",*sum);
fclose(fp1);
sum0=*sum;
delete sum;//////////内存在管理机制为 Cpp 机制
return sum0; ////// Cpp 程序返回计算值
}
```

```
//////////////// 接口程序 mexFunction////////////////////////////////
void mexFunction(int nlhs,mxArray *plhs[],int nrhs, mxArray *prhs[])
{int i;
double *rP,*lP,sum;
mxArray *lP0;
//////////////// 由 prhs[] 中取得 MATLAB 输入数据
rP=new double[nrhs]; /////////内存在管理机制为 Cpp 机制
for(i=0;i<nrhs;i++)rP[i]=mxGetScalar(prhs[i]);
/////////////// 调用 Cpp 程序
sum=OutPut(nrhs,rP);
////////////// 进行数据类型转换，并存入 plhs[] 返回 MATLAB
nlhs=nrhs;
plhs[0]=mxCreateDoubleMATrix(1,nlhs,mxREAL);
lP0=mxCreateDoubleMATrix(1,nlhs,mxREAL);
lP= new double[nlhs];
for(i=0;i<nlhs;i++)lP[i]=i*sum;
for(i=0;i<nlhs;i++)*(mxGetPr(lP0)+i)=*(lP+i);
plhs[0]=mxDuplicateArray(lP0); //使用内存数据复制函数 mxDuplicateArray()
mxDestroyArray(lP0); //////////内存在管理机制为 MATLAB 机制
delete rP;
delete lP;
}
////////////////结束////////////////////////////////////////////////
```

在 MATLAB 命令窗口中将"h1.cpp"文件编译后，得动态链接子程序 h1.dll，并在 MATLAB 命令行中调用。

```
>>mex h1.c
>>a=h1(1,2,3,4,5)
a=
      0    15    30    45    60
```

同时，由于用户 Cpp 程序中执行了文件输出，在当前目录下生成"OutPut1.txt"文件，输出结果为：

```
Input 5 Data is:    1.00    2.00    3.00    4.00    5.00
sum=    15.00
```

以不同参数调用 h1.dll，如

```
>> b=hh1(2,4)
b =
      0    6
```

当前目录下生成"OutPut1.txt"文件，输出结果为：

```
Input 2 Data is:    2.00    4.00
sum=    6.00
```

9.1.2 VC 与 MEX 文件示例二

一般情况下，我们都是在 MATLAB 命令行下编译 MEX 文件，所用命令为：

```
>>mex Filename.c
```

但因 MEX 文件是以 C/C++语言为主编写的程序，只不不过在 MEX 文件中添加了接口函

数 mexFunction，且该函数可与 C/C++函数相一致。因此，可在 VisualC++的 IDE 环境中编译 MEX 文件。这种方法将极大地方便编程者利用 VisualC++ ClassWizard 创建类和维护类，并利用 VisualC++的项目管理进行项目管理。由于此种方法编写的 MEX 文件是在 VisualC++的 IDE 环境中编译，因此，其调试方法与一般 VC 工程基本一致。

本节以"MyDLL"工程为例，阐述在 VisualC++的 IDE 环境中编写和编译 MATLAB MEX 文件。

1. 修改 mex.h

该文件在%MATLAB7%extern\include 目录中，打开该文件，找到 mexFunction 函数申明，利用 VC 中的关键字"__declspec(dllexport)"，申明该函数为导出函数，并将文件另存为"mex_VC.h"。

```
/////////////////原 MEX.h 文件
void mexFunction(
    int          nlhs,            /* number of expected outputs */
    mxArray      *plhs[],         /* array of pointers to output arguments */
    int          nrhs,            /* number of inputs */
    const mxArray *prhs[]         /* array of pointers to input arguments */
);
/////////////////修改并另存为 mex_VC.h
__declspec(dllexport)void mexFunction(
    int          nlhs,            /* number of expected outputs */
    mxArray      *plhs[],         /* array of pointers to output arguments */
    int          nrhs,            /* number of inputs */
    const mxArray *prhs[]         /* array of pointers to input arguments */
);
```

2. 创建 MFC AppWizard(dll)工程 MyDll

运行 VisualC++并单击 File→New，打开 Project 对话框来创建各种 VC 工程，选取 MFC Appwizard(dll)（图 9-1，其他程序基本相同）。在 loctation 编辑框中输入欲创建工程的保存路径，在 Project name 编辑框中输入工程名如"MyDll"，单击 OK 进入下一步工程设置对话框，选中 Regular DLL with MFC Statically Linked，单击 Finish 完成 MyDll 工程创建。

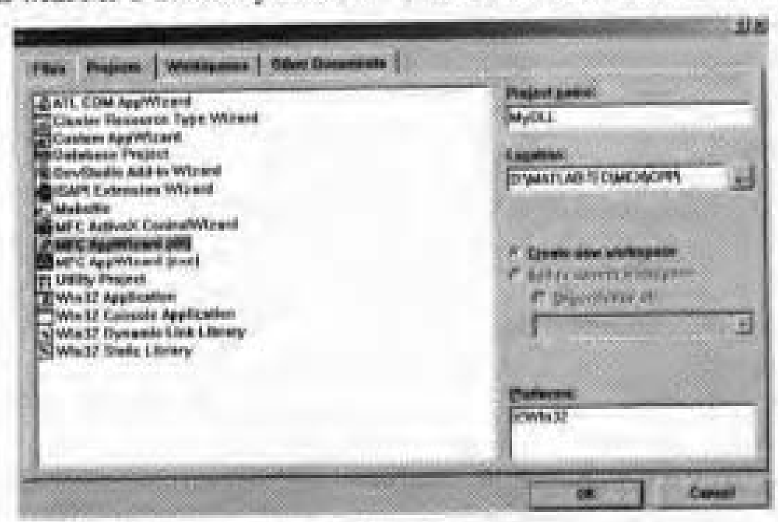


图 9-1　MyDll 工程创建

3．设置系统选项

（1）设定头文件和库文件路径。

菜单 Tools→Options→Directories

Include files:添加%MATLAB7%extern\include

Include files:添加%MICROSOFT VISUALSTUDIO%VC98\MFC\SRC

Library files:添加%MATLAB7%extern\include

（2）设置编译连接选项。

菜单 Project→Settings

C/C++→Preprocessor definitions: 添加 MATLAB_MEX_FILE

Link→Object/Library modules: 添加 libmat.lib libmex.lib libmx.lib

General→Microsoft Foundation Classes: Use MFC in a Static Library

4．编写 DLL 主程序

（1）在 MyDll.h 中添加头文件：

```
#include "mex_vc.h"
```

（2）在 MyDll.cpp 中添加头文件：

```
#include "stdafx.h"
```

（3）编写 mexFunction 函数。

在 MyDll.cpp 文件最后中找到：CMyDllApp theApp;

在其后编写用户 C 函数和 mexFuntion 接口函数如下：

```
//////////////////////////////用户 C 函数, **注：不在 MyDll.h 中进行函数申明
double OutPut(int n, double *rP)
{
 int i;   double *sum,sum0;
 FILE *fp1;

sum=new double[1];////////////////内存在管理机制为 Cpp 机制
if((fp1=fopen("OutPut1.txt","w"))==NULL){printf(" Can't open OutPutfile\n");exit(0);}
*sum=0.0;
fprintf(fp1,"Input  %d Data is:",n);
for(i=0;i<n;i++){*sum=*sum+rP[i];fprintf(fp1,"  %7.2lf ",rP[i]);}
fprintf(fp1,"\nsum=  %7.2lf \n",*sum);
fclose(fp1);
sum0=*sum;
delete sum;/////////内存在管理机制为 Cpp 机制
return sum0; ////// Cpp 程序返回计算值
}

///////////////////////////接口函数, **注：不在 MyDll.h 中进行函数申明
void mexFunction(
int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[]
)//////////////////////////////////////////////注：mexFunction 要与 mex_vc.h 中一致，分行写
{int i;
double *rP,*lP,sum;
mxArray *lP0;
```

```
///////////////// 由 prhs[] 中取得 MATLAB 输入数据
rP=new double[nrhs]; //////////内存在管理机制为 Cpp 机制
for(i=0;i<nrhs;i++)rP[i]=mxGetScalar(prhs[i]);
//////////////// 调用 Cpp 程序
sum=OutPut(nrhs,rP);
/////////////// 进行数据类型转换，并存入 plhs[] 返回 MATLAB
nlhs=nrhs;
plhs[0]=mxCreateDoubleMATrix(1,nlhs,mxREAL);
lP0=mxCreateDoubleMATrix(1,nlhs,mxREAL);
lP= new double[nlhs];
for(i=0;i<nlhs;i++)lP[i]=i*sum;
for(i=0;i<nlhs;i++)*(mxGetPr(lP0)+i)=*(lP+i);
plhs[0]=mxDuplicateArray(lP0); //使用内存数据复制函数 mxDuplicateArray()
mxDestroyArray(lP0); //////////内存在管理机制为 MATLAB 机制
delete rP;
delete lP;
}
///////////////////////结束
```

在 VisualC++中编译得得动态链接子程序 MyDll.dll，并在 MATLAB 命令行中调用。

```
>> clear all
>> Mydll(1,2,4,5,6,7)
ans =
     0    25    50    75   100   125
```

同时，由于用户 Cpp 程序中执行了文件输出，在当前目录下生成“OutPut1.txt”文件，输出结果为：

```
Input 6 Data is:    1.00    2.00    4.00    5.00    6.00    7.00
sum=    25.00
```

9.1.3 VC 与引擎应用程序

在 VisualC++中使用 engine，除 VC 中进行必要的环境配置外，基本上与 C 语言中使用 engine 相一致。本节以一个简单的例子“VCeng1”工程来说明在 VisualC++中使用 MATLAB engine。

1．VisualC++工程创建

启动 VisualC++并单击 File→New，打开 Project 对话框来创建各种 VC 工程，选取 MFC Appwizard(exe)（图 9-2，其他程序基本相同）。在 loctation 编辑框中输入欲创建工程的保存路径，在 Project name 编辑框中输入工程名如“VCeng1”，单击 OK 进入下一步工程设置对话框，选中 single document，单击 Finish 完成 VCeng1 工程创建。

2．环境配置

（1）设定头文件和库文件路径。

菜单 Tools→Options→Directories

Include files:添加%MATLAB7%extern\include

Library files:添加%MATLAB7%\Extern\Lib\Win32\Microsoft\Msvc60

（2）设置编译连接选项。

菜单 Project→Settings

Link→Object/Library modules: 添加 libmat.lib libeng.lib libmx.lib libmex.lib

General→Microsoft Foundation Classes: Use MFC in a Static Library

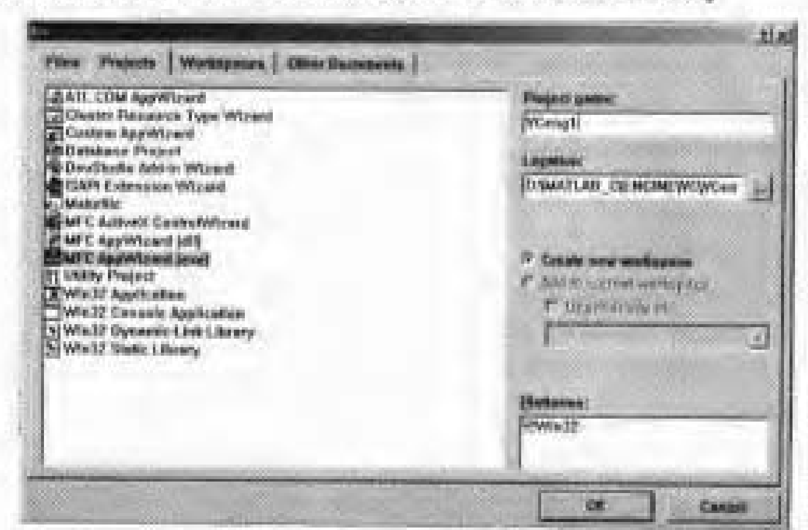


图 9-2 创建 VCeng1 工程

3. 添加响应函数

在 ResourceView 中打开菜单 IDR_MAINFRAME，添加 MATLABEngine→VCeng1 菜单项（图 9-3），再利用 ClassWizard 为 VCeng1 菜单添加 CVCeng1View 窗口 Command 消息映射函数 OnMenuVCeng1()。

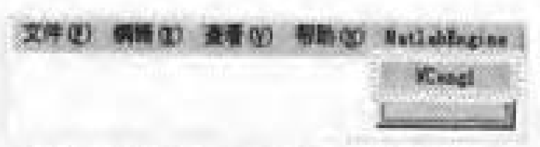


图 9-3 OnMenuVCeng1()消息映射函数

4. 调用引擎库函数

（1）在调用引擎库的头文件中添加引擎库头文件"engine.h"：

本例在"VCeng1View.h"中添加：#include "engine.h"。

（2）在调用引擎库的实现文件中添加引擎调用代码：

本例在"VCeng1View.cpp"的响应函数 OnMenuVCeng1()中添加如下代码：

```
//////////////////////引擎库函数调用代码清单
void CVCeng1View::OnMenuVCeng1()
{
 Engine *ep; /////////引擎对象指针声明
 double t[1];
 t[0]=6.28;
 mxArray *T=NULL;
 if(!(ep=engOpen(NULL)))//////////打开引擎
 { AfxMessageBox("Can not open the MATLAB engine!"); exit(-1);}

 T=mxCreateDoubleMATrix(1,1,mxREAL);
 memcpy((char*)mxGetPr(T),(char*)t,1*sizeof(double));
```

```
////////////////////通过引擎序函数调用 MATLAB 命令
engPutVariable(ep,"ep_T",T);
engEvalString(ep,"ep_T1=0:pi/48:ep_T;");
engEvalString(ep,"ep_D=sin(ep_T1)");
engEvalString(ep,"plot(ep_T1,ep_D,'r-');");

engEvalString(ep,"xlabel('t');");
engEvalString(ep,"ylabel('sin(t)');");
engEvalString(ep,"title('sin(x) curve');");
AfxMessageBox("关闭引擎");
engClose(ep); ////////关闭引擎，引擎工作空间的变量由 MATLAB 自动内存释放机制释放
mxDestroyArray(T); //////Single Document 空间中变量的内存释放
}
///////////////结束
```

在 VisualC++中编译执行 VCeng1，程序通过引擎启动 MATLAB，并调用 MATLAB 命令完成 sin(x)曲线绘制后（图 9-4），调用 MFC 消息框提示用户进程将关闭引擎程序。用户确认后，关闭引擎程序，进程返回到 Single Document 程序框架，以便等待用户的其他操作。

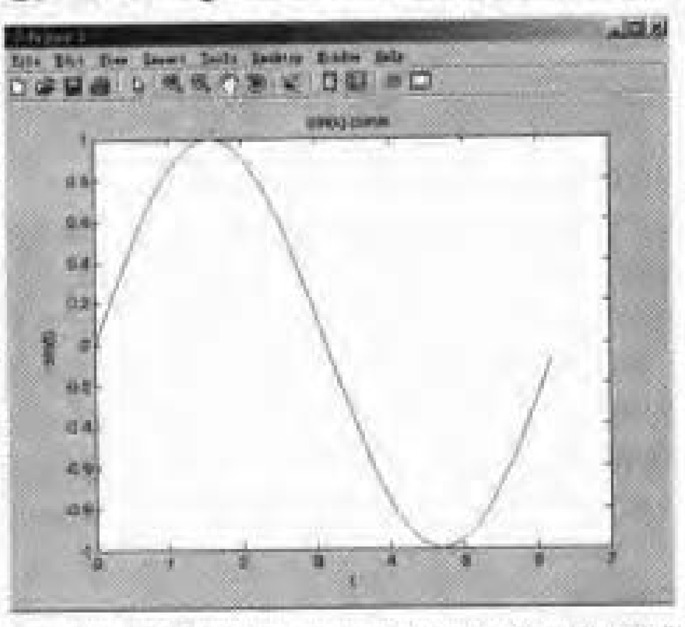


图 9-4　VCeng1 通过引擎库调用 MATLAB 命令绘制的正弦曲线

9.1.4　VC 与 MAT 文件

1. VC 输入输出流与 MAT 文件

Visual C++中数据的输入输出多采用串行化方式进行，串行化是通过 MFC 提供的类 Carchive 对 C++运算符<<和>>重载后进行的。一般情况下，使用 CArchive 对对象进行读操作的过程如下：

```
void CProjectNameDoc::Serialize(CArchive& ar)
{    if (ar.IsStoring()) //////////////写入方式
{
     ar >> obj1>>obj2>>obj3...>>objn; ////按顺序读取数据操作
}
else   ////////////读取方式
```

```
{
    ar << obj1<<obj2<<obj3...<<objn; ////按顺序写入数据操作
}
COleServerDoc::Serialize(ar);// 调用 ColeServerDoc 类实现操作符重载
}
```

MAT 文件是 MATLAB 中数据输入输出的基本方法。在 VisualC++中一般的 MAT 文件读取操作为：

```
mxArray *Read_mxArray(MATFile *pmat,const char *VariableName)
{ mxArray *out; CString str;
out=matGetVariable(pmat, VariableName);
str.Format("读取变量：%s 错误,返回", VariableName); ////定义提示信息
if(out==(mxArray *)NULL){ AfxMessageBox(str); return 0; }
return(out);///////////////////////////返回所读取的变量
}
```

在 VisualC++中一般的 MAT 文件写入操作为：

```
void Write_mxArray(MATFile *pmat, char *VariableName,mxArray *Variable)
{ CString str;
str.Format("写入变量：%s 错误,返回", VariableName); ////定义提示信息
if((matPutVariable(pmat,VariableName,Variable))!=0){AfxMessageBox(str);}
}
```

2. VC 与 MAT 文件示例——VCmat1

从上述两种文件格式的输入输出过程可知，只要把 mxArray 类型的数据 Variable 与 obj 对应起来，并进行相应的数据类型转换，就可以实现 MAT 文件与 Cfile 文件间转换。由于 VisualC++对操作符<<和>>重载后，obj 对象可以是由 Cobject 及其派生类对象，其数据类型远比 mxArray 对象复杂。因此，这种转换方式并不能将所有的 CFile 文件转换为 MAT 文件。此外，在 VisualC++中输入输出 MAT 文件，若仍在 MATLAB 中使用 mex 实现，则与第 7 章用 C 编写 MAT 程序十分相似，若要在 VisualC++的 IDE 中编译，则需进行必要的环境配置。本节以一个简单的例子“VCmat1”工程来说明简单的 CFile 文件在 VisualC++中转换为 MAT 文件的实现。

（1）Visual C++工程创建

启动 Visual C++并单击 File→New，打开 Project 对话框来创建各种 VC 工程，选取 MFC Appwizard(exe)。在 loctation 编辑框中输入欲创建工程的保存路径，在 Project name 编辑框中输入工程名如“VCmat1”，单击 OK 进入下一步工程设置对话框，选中 single document，单击 Finish 完成 VCmat1 工程创建。

（2）添加响应函数

在 ResourceView 中打开菜单 IDR_MAINFRAME，按图 9-5 和表 9-1 添加菜单项和响应函数。

表 9-1　VCmat1 工程菜单项属性与响应函数

菜 单 项	ID	响 应 函 数
SaveAsMATFile	ID_Menu_SaveAsMATFile	void CVCmat1Doc::OnMenuSaveAsMATFile()
InputFromMATFile	ID_Menu_InPutFromMAT	void CVCmat1Doc::OnMenuInPutFromMAT()
MATFileData	ID_Menu_MATFileData	void CVCmat1View::OnMenuMATFileData()
CFileData	ID_Menu_CFileData	void CVCmat1View::OnMenuCFileData()

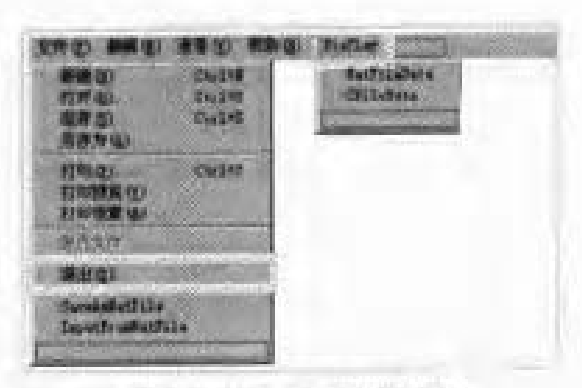


图 9-5　VCmat1 工程菜单项

（3）环境配置

- 设定头文件和库文件路径：

菜单 Tools→Options→Directories

Include files:添加%MATLAB7%extern\include

Library files:添加%MATLAB7%\Extern\Lib\Win32\Microsoft\Msvc60

- 设置编译连接选项：

菜单 Project→Settings

Link→Object/Library modules: 添加 libmat.lib libeng.lib libmx.lib libmex.lib

General→Microsoft Foundation Classes: Use MFC in a Static Library

（4）VCmat1 中添加 MAT 文件读写操作

- 在“VCmat1Doc.h”文件中添加头文件与 mat 文件读写操作函数申明：

```
//////////////////////////////////头文件
#include "mex.h"
#include "mat.h"

//////////////////////////////mat 文件读写操作函数申明
public:
int m,n;
double c_x,*c_A,**c_AB;
mxArray *m_x,*m_A,*m_AB;

mxArray *Read_mxArray(MATFile *pmat,const char *VariableName);
void Write_mxArray(MATFile *pmat, char *VariableName,mxArray *Variable);
```

- 在 CVCmat1Doc()初始化文档时初始化数据：

```
CVCmat1Doc::CVCmat1Doc()
{int i,j;
 m=5; n=10;
 c_x=10.0;/////////变量
 c_A=new double[n]; ////////一维数组
 for(i=0;i<n;i++)c_A[i]=(double)i;
 c_AB=new double*[m];for(i=0;i<m;i++)c_AB[i]=new double[n];//二维数组
 for(i=0;i<m;i++)for(j=0;j<n;j++)c_AB[i][j]=(double)(i*j+1);

 m_x=mxCreateDoubleMATrix(1,1,mxREAL);
 *mxGetPr(m_x)=c_x;
```

```
m_A=mxCreateDoubleMATrix(1,n,mxREAL);
for(i=0;i<n;i++)*(mxGetPr(m_A)+i)=c_A[i];
m_AB=mxCreateDoubleMATrix(m,n,mxREAL);
for(i=0;i<m;i++)for(j=0;j<n;j++){*(mxGetPr(m_AB)+j*m+i)=c_AB[i][j];}
}
```

当然，这些数据也可从其他函数中生成，或从文件中读入。

- 在“VCmat1Doc.cpp”文件中实现 MAT 文件读写操作。

将数据写入 MAT 文件代码：

```
void CVCmat1Doc::OnMenuSaveAsMATFile()
{
MATFile *pmat; CString str;
CFileDialog in(FALSE,NULL,NULL,OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
               "MATLAB Data File(*.mat)|*.mat",NULL);
if(in.DoModal()==IDOK){str=in.GetPathName();
if((pmat=matOpen(str,"w"))==(MATFile *)NULL)///////以写入方式打开 MAT 文件
{AfxMessageBox("Can't open Out MATFile");exit(0);}
Write_mxArray(pmat,"m_x",m_x);////////////////////////////调用写入操作函数，依次写入数据
Write_mxArray(pmat,"m_A",m_A);
Write_mxArray(pmat,"m_AB",m_AB);
matClose(pmat); /////////////////关闭 mat 文件
}
}
////////////////从 MAT 文件读取数据代码：
void CVCmat1Doc::OnMenuInPutFromMAT()
{
MATFile *pmat; CString str;
CFileDialog in(TRUE,NULL,NULL,OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT
               ," MATLAB Data File(*.mat)|*.mat",NULL);
if(in.DoModal()==IDOK){str=in.GetPathName();
if((pmat=matOpen(str,"r"))==(MATFile *)NULL) ///////以读取方式打开 MAT 文件
{AfxMessageBox("Can't open Out MATFile");exit(-1);}
Read_mxArray(pmat,"m_x");////////////////////////////调用读取操作函数，依次读取数据
Read_mxArray(pmat,"m_A");
Read_mxArray(pmat,"m_AB");
matClose(pmat);/////////////////关闭 mat 文件
}
}
//////////////////////////////////读取操作函数
mxArray *CVCmat1Doc::Read_mxArray(MATFile *pmat,const char *VariableName)
{ mxArray *out; CString str;
out=matGetVariable(pmat, VariableName);
str.Format("读取变量：%s 错误,返回", VariableName); ////定义提示信息
if(out==(mxArray *)NULL){ AfxMessageBox(str); return 0; }
return(out);//////////////////////////返回所读取的变量
}
//////////////////////////////////写入操作函数
void CVCmat1Doc::Write_mxArray(MATFile *pmat, char *VariableName,mxArray *Variable)
{ CString str;
str.Format("写入变量：%s 错误,返回", VariableName); ////定义提示信息
if((matPutVariable(pmat,VariableName,Variable))!=0){AfxMessageBox(str);}
}
```

- 在 VCmat1View 窗口中显示从 MAT 文件中读取的数据：

```
void CVCmat1View::OnMenuMATFileData()
{    Invalidate(TRUE);////////////////////重绘窗口
CVCmat1Doc* pDoc=GetDocument();////////取得 Doc 文档句柄
     ASSERT_VALID(pDoc);

     CPaintDC dc(this);///////////////////取得上下文设备
     dc.SetBkMode(TRANSPARENT);          /////设置字体
     dc.SetTextColor(RGB(0,155,155));
LOGFONT logfont;
memset(&logfont,0,sizeof(logfont));
logfont.lfWeight=20;
logfont.lfHeight=20;
lstrcpy(logfont.lfFaceName,"黑体");
CFont nowFont;
nowFont.CreateFontIndirect(&logfont);
     dc.SelectObject(&nowFont);
     ////////////////////////将 mat 文件中取得的数据输出到 view 窗口中
CString str; double *pii; int i,j;
pii=mxGetPr(pDoc->m_x);
     str.Format("Input m_x   :  %7.2f",*pii);  dc.TextOut(10,10,str);

str.Format("Input m_A : ");  dc.TextOut(10,10+logfont.lfHeight,str);
pii=mxGetPr(pDoc->m_A);
for(i=0;i<pDoc->n;i++)
{str.Format("  %7.2f",*(pii+i));
 dc.TextOut(100+i*3*logfont.lfWeight,10+logfont.lfHeight,str);
}

str.Format("Input m_AB: "); dc.TextOut(10,10+2*logfont.lfHeight,str);
     pii=mxGetPr(pDoc->m_AB);
for(i=0;i<pDoc->m;i++)for(j=0;j<pDoc->n;j++)
{str.Format("  %7.2f",*(pii+j*pDoc->m+i));
 dc.TextOut(100+j*3*logfont.lfWeight,10+3*logfont.lfHeight+i*logfont.lfHeight,str);
}
}
```

- 在 VC 串行化读写数据操作：

```
void CVCmat1Doc::Serialize(CArchive& ar)
{int i,j;
if (ar.IsStoring())
{
ar>>c_x;
for(i=0;i<n;i++)ar>>c_A[i];
for(i=0;i<m;i++)for(j=0;j<n;j++)ar>>c_AB[i][j];
}
else
{
ar<<c_x;
for(i=0;i<n;i++)ar<<c_A[i];
```

```
for(i=0;i<m;i++)for(j=0;j<n;j++)ar<<c_AB[i][j];
}
}
```

- 在 VCmat1View 窗口中显示串行化读取的数据：

```
void CVCmat1View::OnMenuCFileData()
{
Invalidate(TRUE);///////////////////重绘窗口
CVCmat1Doc* pDoc=GetDocument();///////取得 Doc 文档句柄
     ASSERT_VALID(pDoc);

     CPaintDC dc(this);////////////////取得上下文设备
     dc.SetBkMode(TRANSPARENT);          /////设置字体
     dc.SetTextColor(RGB(255,0,0));
LOGFONT logfont;
memset(&logfont,0,sizeof(logfont));
logfont.lfWeight=20;
logfont.lfHeight=20;
lstrcpy(logfont.lfFaceName,"隶书");
CFont nowFont;
nowFont.CreateFontIndirect(&logfont);
     dc.SelectObject(&nowFont);
     //////////////////////将 CFile 文件中取得的数据输出到 view 窗口中
CString str; int i,j;
     str.Format("Input c_x   : %7.2f",pDoc->c_x);        dc.TextOut(10,10,str);

str.Format("Input c_A : ");   dc.TextOut(10,10+logfont.lfHeight,str);
for(i=0;i<pDoc->n;i++)
{str.Format("  %7.2f",pDoc->c_A[i]);
 dc.TextOut(100+i*3*logfont.lfWeight,10+logfont.lfHeight,str);
}

str.Format("Input c_AB: ");  dc.TextOut(10,10+2*logfont.lfHeight,str);
for(i=0;i<pDoc->m;i++)for(j=0;j<pDoc->n;j++)
{str.Format("  %7.2f",pDoc->c_AB[i][j]);
 dc.TextOut(100+j*3*logfont.lfWeight,10+3*logfont.lfHeight+i*logfont.lfHeight,str);
}
}
```

在 Visual C++中编译执行 VCmat1 工程，从 MAT 读取数据在 View 窗口中显示结果如图 9-6 所示，由串行化读取数据在 View 窗口中显示结果如图 9-7 所示。从中可知，VCmat1 工程实现了 MAT 文件的读写操作。

图 9-6　从 MAT 读取数据在 View 窗口中显示结果

图 9-7　串行化读取数据在 View 窗口中显示结果

9.2　MCC

MCC 是 MATLAB 中经过优化的编译器。使用 MCC，用户可将 MATLAB 数学库、图形库和界面的 MATLAB 程序转化为独立于 MATLAB 的 EXE 应用程序和 DLL 动态链接库。结合 MATLAB 建立 VC 独立应用程序的实现步骤为：

（1）编写 m 文件；

（2）配置 MATLAB 编译器，并使用 mcc 编译 m 文件；

（3）创建 Visaul C++工程；

（4）使用由 mcc 编译生成的 cpp 文件。

9.2.1　准备工作

1．mcc 编译选项说明

MATLAB7 版本对 mcc 编译设置有较大改动。尤其是将 m 文件编译为“.h”和“.c”文件及相应的动态链接库时，7.0 以前版本主要使用：

```
>>mcc –t –L Cpp filename1, filename2,……
```

而在 MATLAB7 版本中，主要使用：

```
>>mcc -B csharedlib:libFileName FileName1.m FileName2.m … -v
```

所生成的文件也有一定差别，MATLAB 7 版本中一般可生成 8 个文件：

filename.h——库包装文件的头文件。

filename.c——库包装文件的实现部分。

filename _mcc_component_data.c——初始化文件。

filename.exports——函数库的导出列表。

filename.lib——生成的库文件。

filename.dll——生成的动态链接库。

filename.exp——386 模式执行程序。

filename.ctf——包装文件信息。

2．Visual C++环境配置方法一

Visual C++环境配置最主要的是包含文件路径和链接库及其路径设置。此外，由于用户 m 文件不同，所生成的 MATLAB 组件运行时（MCR）可能有不同，相应的 VC 工程中设置也会有一定的差异。因此，用户在设置 VC 环境配置时，应搜索需添加的包含文件路径（include directories）、链接库路径（linker library directories）和 MATLAB 组件运行时（MCR）。搜索方

法就是在 mcc 编译选项中添加“–v”选项。下面以“mcc2”工程为例进行阐述。

9.2.2 建立独立应用程序示例

1. 编写 m 文件“test.m”

```
function [a,b]=test(x,y)
a=x+y;
b=x.*y;
subplot(1,2,1);plot(x,a); xlabel('X');ylabel('X+Y');
subplot(1,2,2);plot(x,b); xlabel('X');ylabel('X.*Y');
```

2. 配置 MATLAB 编译器，并使用带“–v”选项的 mcc 命令编译 m 文件“test.m”

（1）用“mbuild –setup”将 MATLAB 编译器设置为 VisualC++6.0

```
>> mbuild -setup
Please choose your compiler for building standalone MATLAB applications:
Would you like mbuild to locate installed compilers [y]/n? y
Select a compiler:
[1] Lcc C version 2.4 in C:\MATLAB7\sys\lcc
[2] Microsoft VisualC/C++ version 6.0 in C:\Microsoft VisualStudio
[0] None
Compiler: 2
Please verify your choices:
Compiler: Microsoft VisualC/C++ 6.0
Location: C:\Microsoft VisualStudio
Are these correct?([y]/n): y
Warning: Mbuild requires that the Microsoft VisualC++ 6.0
directories "VC98" and "Common" be located within the same parent directory.
(Expected to find a directory named "Common" in the directory 'C:\Microsoft VisualStudio'.)
Try to update options file: C:\Documents and Settings\Administrator\Application Data\MAThWorks\
    MATLAB\R14\compopts.bat
From template:              C:\MATLAB7\BIN\WIN32\mbuildopts\msvc60compp.bat
Done . . .
--> "C:\MATLAB7\bin\win32\mwregsvr C:\MATLAB7\bin\win32\mwcomutil.dll"
DllRegisterServer in C:\MATLAB7\bin\win32\mwcomutil.dll succeeded
--> "C:\MATLAB7\bin\win32\mwregsvr C:\MATLAB7\bin\win32\mwcommgr.dll"
DllRegisterServer in C:\MATLAB7\bin\win32\mwcommgr.dll succeeded
```

至此，完成 MATLAB 编译器设置。

（2）以带“–v”选项的 mcc 命令编译 m 文件“test.m”

```
>> mcc –B csharedlib:libtest test.m –v
```

其中，操作参数–Bcsharedlib 是一个绑定的操作，其等效指令为–W lib:<libname>–T link:lib。–v 选项显示编译信息。在众多的编译信息中，可确定 Visual C++包含文件路径（include directories）和链接库路径（linker library directories）配置。

（3）包含文件路径（include directories）

包含文件路径是以“C1”开头且提示符为"–I"的路径，本例中以"C1"开头有两行，且提示符为"–I"的路径均一致：

```
–IC:\MATLAB7\extern\include –IC:\MATLAB7\simulink\include –O2 –DNDEBUG
```

–IC:\MATLAB7\extern\include –IC:\MATLAB7\simulink\include –O2 –DNDEBUG

其中：–IC:\MATLAB7\simulink\include -O2 -DNDEBUG 为 DNDEBUG 路径，不需在 VC 工程中添加，故 VC 中需添加的包含文件路径（include directories）为：

C:\MATLAB7\extern\include

（4）链接库路径（linker library directories）

链接库路径是以"link"且提示符为"/LIBPATH:"的路径，本例中为：

/LIBPATH:"C:\MATLAB7\extern\lib\win32\microsoft\msvc60"

故 VC 工程中需添加的链接库路径为：

C:\MATLAB7\extern\lib\win32\microsoft\msvc60

（5）MATLAB 组件运行时（MCR）

MATLAB 组件运行时（MCR）是以"link"且提示符为"/nologo"的文件，本例中为：

mclmcrrt.lib 和 libtest.lib

3. 创建 Visaul C++工程 mcc2

运行 Visual C++并单击 File→New，打开 Project 对话框来创建各种 VC 工程，选取 MFC Appwizard(exe)（图 9-8，其他程序基本相同）。在 loctation 编辑框中输入欲创建工程的保存路径，在 Project name 编辑框中输入工程名如 "mcc2"，单击 "OK" 进入下一步工程设置对话框，选中 Dailog Based，单击 Finish 完成 mcc2 工程创建。

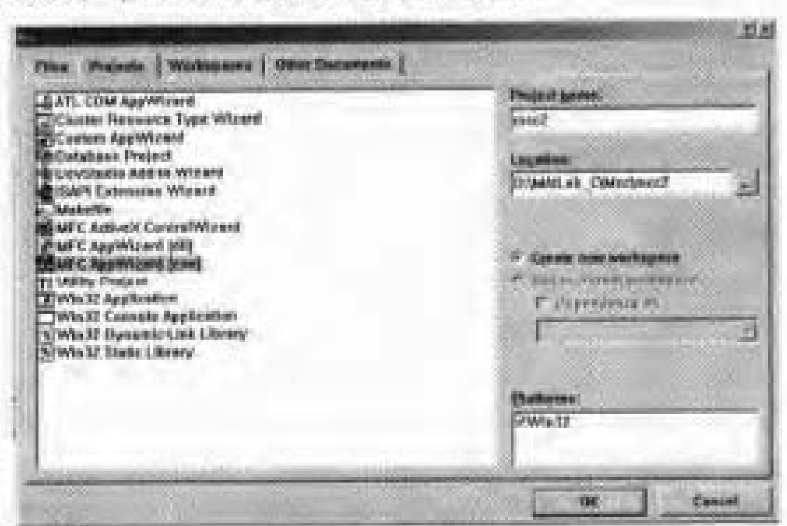


图 9-8　mcc2 工程创建

4. VisualC++环境配置

（1）设定头文件和库文件路径

菜单 Tools→Options→Directories

Include files:添加%MATLAB7%extern\include

Library files:添加%MATLAB7%extern\include

（2）设置编译连接选项

菜单 Project→Settings

Link→Object/Library modules: 添加 mclmcrrt.lib libtest.lib

General→Microsoft Foundation Classes: Use MFC in a Static Library

5. 编写 mcc2 工程对话框

创建如图 9-9 所示 mcc2 工程对话框。对话框控件属性如表 9-2 所示。利用 ClassWizard 分别为 IDC_EDIT_Xydim 编辑框、IDC_LIST_Out 和 IDC_LIST_Out2 列表控件添加变量 “m_XYdim”、“m_Mcc2_List” 和 “m_Mcc1_List”，为 IDC_BUTTON_Mcc1 按钮添加响应函数 “OnBUTTONMcc1()”。

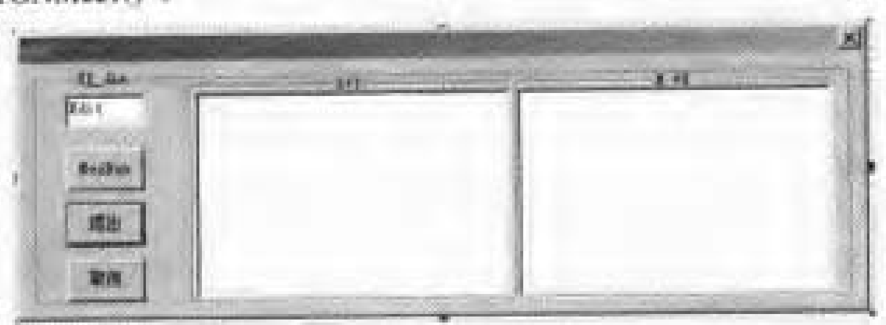


图 9-9　创建 mcc2 工程对话

表 9-2　mcc2 对话框控件属性

控件属性	控制 ID	说　明
Edit Box	IDC_EDIT_XYdim	接受用户输入 X 和 Y 向量的维数
Button	IDC_BUTTON_Mcc1	鼠标左键单击该控件，调用 libtest 中对应于 test.m 函数
Button	IDOK	结束对话框
Button	IDCANCEL	取消对话框进程
List Box	IDC_LIST_Out	用于显示 X+Y 计算结果
List Box	IDC_LIST_Out2	用于显示 X.*Y 计算结果

6. 添加 mcc 编译文件

将 mcc 编译 “test.m” 生成的 “libtest.h”、“libtest.lib”、“libtest.dll” 和 “libtest.ctf” 4 个文件复制至 mcc2 工程目录中。并用 project→add to project→Files 将 “libtest.h” 加入 mcc2 工程中。在对话框程序文件 “mcc2Dlg.h” 中添加头文件 “libtest.h” 和 “mclmcr.h”。

打开 “libtest.h”，从程序清单中可知，libtest.h 文件中主要声明程序运行时所需的初始化函数 libtestInitialize()、程序终止时 libtestTerminate()函数，以及对应 test.m 文件中 mlfTest()函数。其中 mlfTest()参数为：输出参数个 nargout；输出参数列表：mxArray** a, mxArray** b，分别对应 test.m 文件中的左手侧参数；输入参数列表：mxArray* x, mxArray* y，分别对应 test.m 文件中的右手侧参数。因此，用 mcc 编译器将 m 文件转化为 C 程序，其数据类型是 mxArray 类对象。

```
//////////////////////////////// libtest.h 文件清单
#ifndef __libtest_h
#define __libtest_h 1
#if defined(__cplusplus) && !defined(mclmcr_h) && defined(__linux__)
#   pragma implementation "mclmcr.h"
#endif
#include "mclmcr.h"
#ifdef __cplusplus
extern "C" {
#endif
```

```
extern bool libtestInitializeWithHandlers(mclOutputHandlerFcn error_handler,
                                          mclOutputHandlerFcn print_handler);
extern bool libtestInitialize(void);      ///////////////初始化函数
extern void libtestTerminate(void);       ///////////////终止函数
extern void mlxTest(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[]);///默认函数
extern void mlfTest(int nargout, mxArray** a, mxArray** b
                    , mxArray* x, mxArray* y);//////////对应于 test.m 中函数
#ifdef __cplusplus
}
#endif
#endif
```

7. 初始化进程

在 BOOL CMcc2Dlg::OnInitDialog()函数中添加以下代码，以便在进行对话框初始化后，进行初始化 libtest.dll 进程，代码置于对话框初始化与进程返回之间。

```
if( !mclInitializeApplication(NULL,0) )
{AfxMessageBox( "Could not initialize the application."); exit(1); }
    if (!libtestInitialize())
    {AfxMessageBox("Could not initialize the library."); exit(1); }
```

8. 调用 mlfTest()函数

在 void CMcc2Dlg::OnBUTTONMcc1()函数中添加以下代码，实现对 test.m 调用。

```
void CMcc2Dlg::OnBUTTONMcc1()
{
CString str1,str2;
mxArray *in1, *in2; ///////输入参数，分别对应 test.m 中右手侧参数 x,y
mxArray *out1 = NULL; ///////输出参数，分别对应 test.m 中右手侧参数 a,b
mxArray *out2 = NULL;
double *data;
int i,n;
UpdateData(TRUE); n=m_XYdim;///将编辑框数据传送机制设为从编辑框到变量
data=new double[n];
for(i=0;i<n;i++)data[i]=(double)i;
in1 = mxCreateDoubleMATrix(1,n,mxREAL); ////////mxArray 对象分配地址
in2 = mxCreateDoubleMATrix(1,n,mxREAL);
memcpy(mxGetPr(in1), data, n*sizeof(double)); ///将 C 数据类型转换为 mxArray 数据
memcpy(mxGetPr(in2), data, n*sizeof(double));
mlfTest(2,&out1,&out2,in1,in2);///////////////调用 libtest.h 中函数，对应于 test.m

for(i=0;i<=n;i++){
str1.Format("%8.2f +  %8.2f =  %8.2f   ",
        *(mxGetPr(in1)+i),*(mxGetPr(in2)+i),*(mxGetPr(out1)+i));
str2.Format("%8.2f .*  %8.2f =  %8.2f   ",
        *(mxGetPr(in1)+i),*(mxGetPr(in2)+i),*(mxGetPr(out2)+i));
m_Mcc1_List.AddString(str1); ///用列表控件将计算结果显示出来
m_Mcc2_List.AddString(str2);
}
mxDestroyArray(out1); mxDestroyArray(in1); ///内存释放
mxDestroyArray(out2); mxDestroyArray(in2);
delete data;
}
```

9. 进程终止与资源释放

利用 ClassWizard 为对话框添加 void CMcc2Dlg::OnDestroy()函数，并添加以下代码。

```
void CMcc2Dlg::OnDestroy()
{
CDialog::OnDestroy(); //////结束对话框并释放资源
libtestTerminate();
mclTerminateApplication();
}
```

10. 编译运行

在本机上编译运行后结果如图 9-10 和图 9-11 所示。

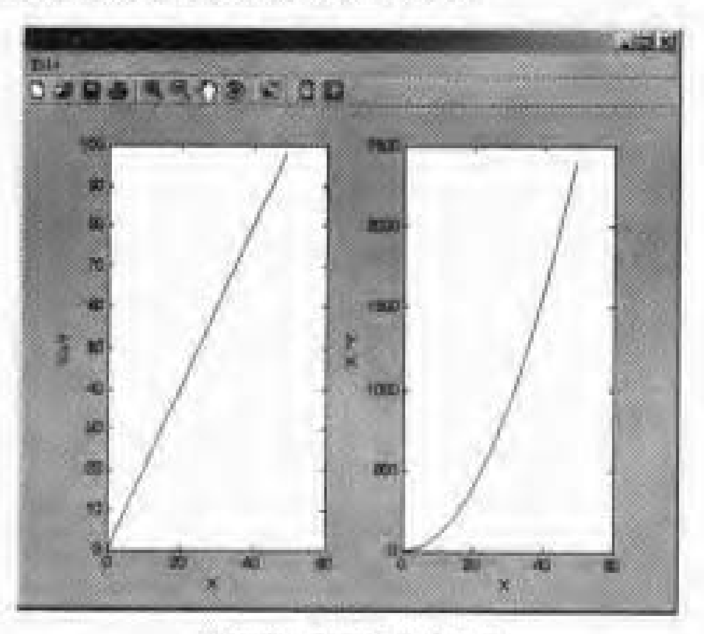


图 9-10 Mcc2 运行结果 1

图 9-11 Mcc2 运行结果 2

11. 独立应用程序发布

首先，将 MATLAB6p5\extern\lib\win32\MCRInstaller.exe 复制到目标计算机，并在目标机器上运行，解压时注意要保持解压后的路径与第一台计算机上 MATLAB 的安装路径一致，安装好 MCR 之后，将<mcr_root>\runtime\win32 加入到系统的环境变量 path 中去。

其次，将工程文件的可执行程序(exe)，共享库(DLL)，共享库对应的.ctf 文件复制到目标计算机。

9.3 MATcom 与 Add-in

MATcom 是 MAThworks 公司推出的第一个由 MATLAB 到 C++的编译开发软件平台，其最后版本为 MATcom 4.5，其集成调试编译环境为 MIDEVA。通过 MATcom 连接 MATLAB m 文件有以下三种方法。

（1）经过简单设置后，由 MIDEVA 将 m 源文件转换为 C/C++，然后添加到 MSVC 工程中。

（2）由 MIDEVA 直接生成 EXE 文件，然后在 VC 中通过 Shell 调用。这种方法简单方便，但运行时出现一个控制台窗口，而且由于 VC 和 MATLAB 之间不能交互，适用性差，仅适用于 VC 中调用 MATLAB 实现图形显示的场合。

（3）通过 Visual MATcom 工具条，使用 Add-in，这种方法提供了一个 MATLAB 和 VC 直接集成的途径，且可快速集成 m 文件到 VC 工程中创建独立的 C/C++应用程序、C MEX DLL，在调试过程中可以查看矩阵变量的值，可直接修改 m 源文件而不是修改生成的 C/C++ 文件。因此，本节主要介绍 Add-in 方法的实现。

9.3.1 MATcom 安装与生成 Visual MATcom 工具条

采用 Visual MATcom 方式，在脱离 MATLAB 环境条件下实现 VC++对 m 文件调用，将大大方便用户应用程序的开发。其实现过程是首先安装 MATcom 4.5 应用程序，然后在 Visual C++ 中设置 MATcom 调用环境，并生成 Visual MATcom 工具条。

（1）在安装 MATcom 4.5 后，在第一次运行时，MATcom 4.5 自动搜索 VC 编译器并提示用户是否安装（图 9-12），用户回答接受安装后，提示是否安装有 MATLAB（图 9-13），回答安装后，MATcom 4.5 往往发生错误，原因是在%MATLAB7%\bin 中没有相应的目录。这时，用户可以在%MATLAB7%\bin 目录下手动建立\toolbox\MATLAB\general 目录，再重新启动 MATcom 4.5 即可。MATcom 4.5 安装成功后，出现图 9-14 所示消息框。其实是否安装有 MATLAB 对 MATcom 没有什么影响，你完全可以选择没有安装 MATLAB，仍然可以编译大多数文件。需要 MATLAB\toolbox 下的文件时，用 addpath()添加路径或者复制到当前目录下就可以了。

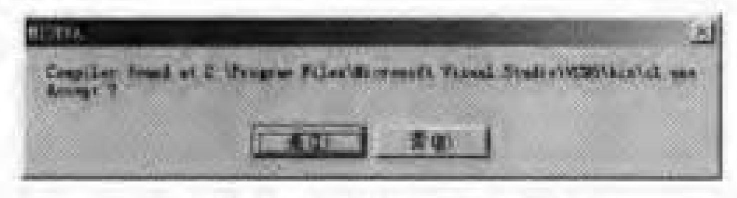


图 9-12 第一次运行 MATcom 4.5 提示一



图 9-13 第一次运行 MATcom 4.5 提示二

（2）复制%MATcom45%\bin\usertype.dat 文件（%MATcom45%指 MATcom 的安装路径）到%Visualc++%\Common\MSDev98\Bin 目录（%Visualc++%指 VC 的安装路径）下。

（3）运行 Visual C++，并从菜单条中选择 Tools→Customize→Add-ins and Macro Files，选择 Browse，改变文件类型为 Add-ins(.dll)，选定%MATcom45%\bin\mvcide.dll 文件，确定。

（4）在 Visual C++的开发环境中可以看到一个如图 9-15 所示的 Visual MATcom 工具条，表明安装成功。可以开始 MATcom 文件编写与调用。



图 9-14　MATcom 4.5 安装成功后的消息框



图 9-15　Visual MATcom 工具条

9.3.2　m 文件转换示例——Test1

本节以一个线性方程组的求解过程为例，介绍通过 MATcom 实现 VC 对 MATLAB m 文件的调用。

（1）编写 MATLAB m 文件 “LineEq.m”，用于求解线性方程组 *AX=B* 的解。

MATLAB m 文件清单：

```
%Slove linearEquation: A*X=B
function X = LineEq (A, B)
X = A\B;
```

（2）创建 VC 工程 “Test1”。

运行 Visual C++并单击 File→New，打开 Project 对话框来创建各种 VC 工程。这里以较为简单的控制台程序为例，选取 Win32 Console Application（图 9-16，其他程序基本相同）。在 loctation 编辑框中输入欲创建工程的保存路径，在 Project name 编辑框中输入工程名如“Test1”，单击 “OK” 进入下一步工程设置对话框，选取 “A single application”，单击 “Finish” 完成工程创建工作。

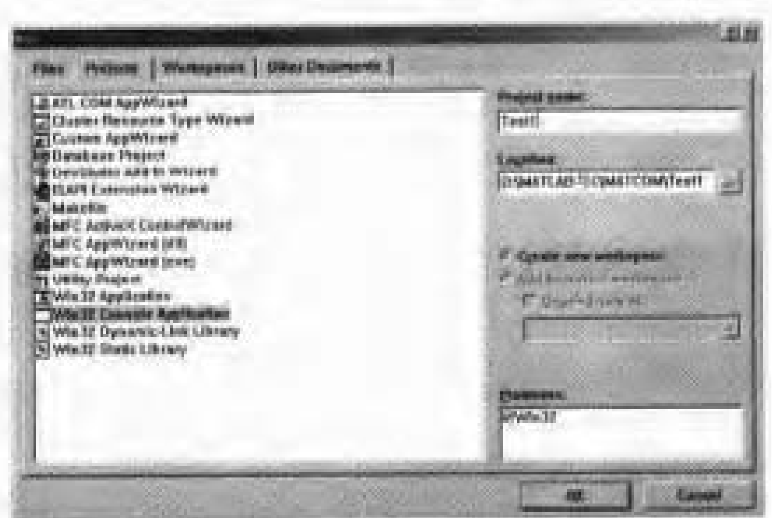


图 9-16　创建 Test1 工程

（3）转换 m 文件 “LineEq.m”。

单击 Visual MATcom 工具栏上的 “m++” 图标，选择保存过的 MATLAB 文件 LineEq.m 进行转化。如果看到的转化信息提示没有错误就可以观察到此时在 FileView 标签中多了 m-files、C++files created from m-files、MATrix<lib>和 External Dependencies 等文件夹（图 9-17）。在相应的文件夹下增加了 LineEq.m、LineEq.cpp、v4501v.lib、LineEq.h 和 matlib.h 等文件。表

明 m 文件转换成功；否则，表明 m 文件中存在错误，可双击 m-files\LineEq.m 进行修改，再重新转化直到没有错误报告为止。

LineEq.m 经转换后所生成的 LineEq.h 和 LineEq.cpp 文件，分别进行了 LineEq 函数（以 m 文件名为函数名，建议用户不修改此函数名，若必须修改时，要注意函数名在相应文件中的统一）和函数体的定义。打开 LineEq.h，从程序清单中可知 LineEq 函数有两个参数，分别为 A 和 B，与 LineEq.m 文件中的输入参数 A 和 B 相对应，其数据类型为 Mm。Mm 数据类型是 MAThtools 公司利用 MATcom 技术开发的矩阵数学库 matrix<lib>中定义的基本数据类型，是一种双精度的 matrix 数据，它可以是复数矩阵、实数矩阵、稀疏矩阵甚至 N 维矩阵；LineEq 函数返回值也为 Mm 类型。



图 9-17　LineEq.m 文件转换结果

```
//////////////////////////////////////////  LineEq.h 清单
#ifndef __LineEq_h
#define __LineEq_h

Mm LineEq(Mm A, Mm B);

#endif // __LineEq_h
/////////////  结束
```

打开 LineEq.cpp，从程序清单中可知，LineEq.cpp 包括两部分：头文件中包含了"matlib.h"、hdrstop 和"LineEq.h"三个文件。其中"matlib.h"将 MATcom 数学库包含至文件中，在使用此包含语句前，须按 9.3.1 中方法在 Visual C++中设置 MATcom 调用环境：

```
pragma hdrstop
```

"LineEq.h"作用与一般的 C++程序相同，是函数定义头文件，为程序提供转换后 C++代码中使用的数据类型、函数原型及常量等信息。

```
//////////////////////////////////////////////////////// LineEq.cpp 清单
  //////////////////////  头文件
#include "matlib.h"
  #pragma hdrstop
  #include "LineEq.h"
  ////////////////////// LineEq 函数体
  Mm LineEq(Mm A, Mm B) {
    begin_scope
    A.setname("A"); B.setname("B");
    dMm(X);

    #line 1 "d:/MATLAB_C/MATcom/lineeq.m"
    call_stack_begin;
    #line 1 "d:/MATLAB_C/MATcom/lineeq.m"
    // nargin, nargout entry code
    double old_nargin=nargin_val; if (!nargin_set) nargin_val=2.0;
    nargin_set=0;
    double old_nargout=nargout_val; if (!nargout_set) nargout_val=1.0;
```

```
        nargout_set=0;

        // translated code

        #line 2 "d:/MATLAB_C/MATcom/lineeq.m"
        _X = mldivide(A,B);
        call_stack_end;

        // nargin, nargout exit code
        nargin_val=old_nargin; nargout_val=old_nargout;

        // function exit code
        A.setname(NULL); B.setname(NULL);
        return X;
        end_scope
      }
///////////////////////////////////// 结束//////////////////////////////
```

（4）在 VC 工程“Test1”中调用转换后函数“LineEq”。

m 文件经转换后所得的*.h 和*.cpp 文件，与一般的 C 语言编写的程序文件一样，可在各种 VisualC++工程中调用。“Test1”工程调用“LineEq”程序清单如下。

```
////////////////////////////////////////////////////////// 头文件
#include "stdafx.h"
#include "stdio.h"
#include "matlib.h" /////包含 MATcom4.5 数学库
#include "LineEq.h"

int main(int argc, char* argv[])
{
//////////////////////////解线性方程组：
 initM(MATCOM_VERSION); ///初始化 matlib 库
 Mm a,b,x; ////使用矩阵类 Mm 构造矩阵 a,b,x.
 a = (BR(1),2,3,semi,4,5,6,semi,7,8,1);
///////给矩阵 a 赋值，BR 是 MATrix<LIB>库的一个宏，用于定义一个矩阵的开始；
///////semi 是 matrix<Lib>库中的一个常量，作用与“;”相当。
 b = zeros(3,1); ///初始化矩阵 b 为零矩阵 3 行 1 列
 b(1,1) = 37; b(2,1) = 85; b(3,1) = 69; ///给矩阵 b 赋值
 x = LineEq(a,b); ///调用转化的函数，求解线性方程组的解
 for (int i = 1; i <= x.rows(); i++) ///把解矩阵 X 的元素显示出来
 {
 for (int j=1;j<=x.cols();j++)
 printf("x(%d,%d)=%f\n",i,j,x.r(i,j));
 }
 exitM(); ///结束对 matlib 库的调用
return 0;
}
//////////////////////////////////////////////结束//////////////////////////////////
```

从程序清单中可知，对“LineEq”调用步骤为：

① 头文件中需包含"matlib.h"文件，以提供 MATcom4.5 数学库；

② 初始化 MATrix<Lib>:

初始化函数为：initM(MATCOM_VERSION);

该函数可多次调用，参数 MATCOM_VERSION 是一个在 matlib.h 中定义了的常量，它保证了动态链接库与 matlib.h 相匹配；

③ 转换函数调用：M 文件经转换后生成的函数，其参数和返回值均为 MATcom 中的 Mm 数据类型。因此，转换函数的调用，其语法格式虽与一般 C 程序调用相同，但对函数参数及返回值的赋值、存取等操作中，应注意 Mm 数据类型的操作方法，Mm 对象操作见 9.3.4 见。

④ 结束 MATrix<Lib>类库：

结束函数为：exitM();

Test1 工程的编译、链接与一般 C++程序一样。编译链接成功后，得独立应用程序"Test1.exe"，执行程序后，得线性方程组的解为：

x(1,1) = 3.000000

x(2,1) = 5.000000

x(3,1) = 8.000000

9.3.3 m 文件转换示例——Test2

"Test2"是以 MFC 创建的对话框工程，在该对话框中调用经转换后的 m 文件以完成正弦函数和余弦函数曲线绘制，其实现步骤如下。

1. 编写 m 文件"Mysin.m"和"Mycos.m"

```
%////////// Mysin.m 程序清单///////////
function Mysin(x1,x2)
X=x1:0.1:x2;
Y=sin(X);
plot(X,Y);
%////////// Mycos.m 程序清单///////////
function Mycos(x1,x2)
X=x1:0.1:x2;
Y=cos(X);
plot(X,Y);
/////////////////////////////////////////
```

2. 创建 VC 工程"Test2"

运行 Visual C++并单击 File→New，打开 Project 对话框，创建 Test2 工程。选取 MFC AppWizard (exe)（图 9-18）。在 loctation 编辑框中输入欲创建工程的保存路径，在 Project name 编辑框中输入工程名如"Test2"，单击"OK"进入下一步工程设置对话框，选取"Dialog based"，单击 Finish 完成"Test2"工程创建。

3. 转换 m 文件"Mysin.m"和"Mycos.m"

单击 Visual MATcom 工具栏上的 m++图标，选择保存过的 MATLAB 文件 Mysin.m 进行转换，完成后再次单击 m++图标，选择 Mycos.m 转换（如果有多个 m 文件，可逐个添加并转换）。如果看到的转化信息提示没有错误就可以观察到此时在 FileView 标签中多了 m-files、C++files created from m-files、MATrix<lib>和 External Dependencies 等文件夹（图 9-19）。在相应的文件夹下增加了 Mycos.m、Mysin.m、Mycos.cpp、Mysin.cpp、v4501v.lib、Mycos.h、Mysin.h

和 matlib.h 等文件，表明 m 文件转换成功。分别在“Mycos.h”和“Mysin.h”文件中添加头文件“matlib.h”，进行编译，若编译成功，进入下一步；否则，表明 m 文件中存在错误，可双击 m-files 目录下相应的 m 文件进行修改，并重新转换直到没有错误报告为止。

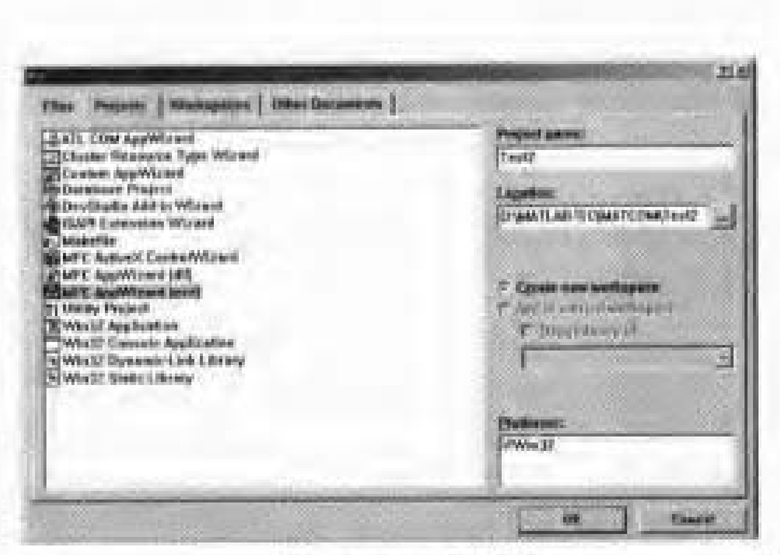


图 9-18　Test2 工程创建

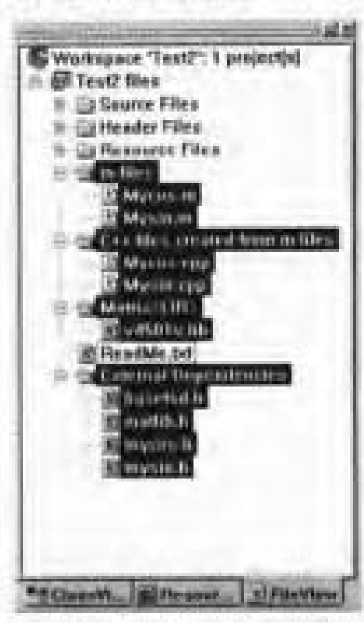


图 9-19　Mysin.m 和 Mycos.m 文件转换结果

```
//////////   Mycos.h   文件清单
#include "matlib.h" //////////////需添加"matlib.h"头文件，否则，编译时报错
#ifndef __Mycos_h
#define __Mycos_h

Mm Mycos(Mm x1, Mm x2);

#endif // __Mycos_h

//////////   Mysin.h   文件清单

#include "matlib.h" //////////////需添加"matlib.h"头文件，否则，编译时报错
#ifndef __Mysin_h
#define __Mysin_h

Mm Mysin(Mm x1, Mm x2);

#endif // __Mysin_h
```

4. 在 VC 工程“Test2”中调用转换后函数“Mysin”和“Mycos”

（1）创建如图 9-20 所示对话框界面，并按表 9-3 设置各控件属性。

表 9-3　Test2 对话框控件属性

控件属性	控制ID	说　明
Edit Box	IDC_EDIT_x1	接受用户输入的左端点 x1，与之相连的变量为 m_x1
Edit Box	IDC_EDIT_x2	接受用户输入的左端点 x2，与之相连的变量为 m_x2
Button	IDC_BUTTON_MySin	鼠标左键单击该控件，调用 Mysin 函数并绘制正弦曲线

续表

控件属性	控制ID	说明
Button	IDC_BUTTON_MyCos	鼠标左键单击该控件，调用 Mycos 函数并绘制余弦曲线
Static Text	IDC_Statistic_MySin	用于显示正弦曲线
Static Text	IDC_Statistic_MyCos	用于显示正弦曲线



图 9-20　Test2 对话框

（2）添加头文件：

```
#include "Mysin.h"
#include "Mycos.h"
#include "matlib.h"
```

（3）为 IDC_BUTTON_MySin 添加响应代码：

```
void CTest2Dlg::OnBUTTONMySin()
{
double x1,x2;
UpdateData(TRUE); ////////////////////// 将编辑框中输入值赋给与之相连的变量
 x1=(double)m_x1;  x2=(double)m_x2;
CWnd* psin=(CWnd*)GetDlgItem(IDC_Statistic_MySin);
 Mm plothandle=winaxes(psin->m_hWnd); /////////读取图形句柄
 initM(MATCOM_VERSION); //////////初始化 matlib 库
Mysin(x1,x2); ////////////// 调用转换生成的函数
 exitM();    ////////////////////// 退出 matlib 库
}
```

（4）为 IDC_BUTTON_MyCos 添加响应代码：

```
void CTest2Dlg::OnBUTTONMyCos()
{
double x1,x2;
UpdateData(TRUE);    ////////////////////// 将编辑框中输入值赋给与之相连的变量
 x1=(double)m_x1;  x2=(double)m_x2;
CWnd* pcos=(CWnd*)GetDlgItem(IDC_Statistic_MyCos);
 Mm plothandle=winaxes(pcos->m_hWnd);    /////////读取图形句柄
initM(MATCOM_VERSION);    //////////初始化 matlib 库
 Mycos(x1,x2);    ////////////// 调用转换生成的函数
 exitM();  ////////////////////// 退出 matlib 库
}
```

（5）编译链接 Test2 工程：

按一般的 C++程序进行编译链接 Test2 工程，这其中，由于 Mysin.h 和 Mycos.h 函数定义文件是相应的 m 文件转换生成的，生新编译整个工程时，这两个文件同时重新转换，VC++询问用户外部 Mysin.h 和 Mycos.h 已发生修改，是否重新加载这两个文件时，一般选择不重新加载，因重新转换生成的 Mysin.h 和 Mycos.h 文件中，没有包含“matlib.h”头文件，需用户手动添加，若没有包含该文件，编译器不能识别 Mm 数据类型而发生错误。

（6）执行 Test2.exe：

执行 Test2.exe 文件后，得如图 9-21 所示对话框，在编辑框中输入 0 和 25（4π 近似值），单击 sin 和 cos 按钮后，在左框中绘制了相应的曲线。

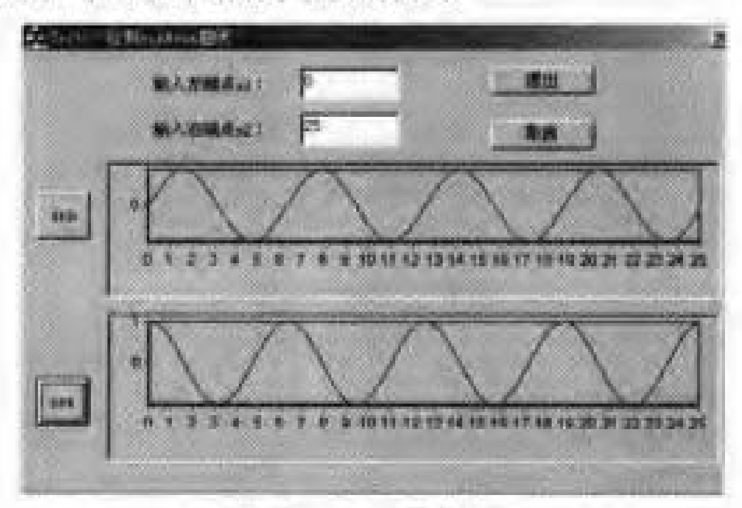


图 9-21　Test2 运行结果

5．独立应用程序发布

通过 MATcom 把.m 文件编译为为 stand_alone 的程序，一般不需要 MATLAB 系统，仅需复制一些必要的 dll 文件，这些文件主要有：v4501v.dll、ago4501.dll、icudt24l.dll 以及%MATLAB%\bin\win32\目录中以 lib 开头的 dll 文件，当然，并不是每个独立应用程序都需要所有以 lib 开头的 dll 文件，只是将这些 lib*.dll 文件都复制后，较为方便。

9.3.4　matlib 数学库与 Mm 数据类型

从 Test1 和 Test2 工程中可知，MATcom 与 VC++中数据交换，是基于 MAThtools 提供的 MATcom C 数学库函数及其数据类型 Mm 的。因此，本节对 MATcom C 数学库函数及其数据类型 Mm 作简单介绍，详细情况可查阅%MATcom4.5%\lib\matlib.h 文件。

MATcom C 数学库函数是一个丰富的数学库，它提供的数学函数约 600 个，这些数学函数可分为：矩阵基础类、系统常数、特殊函数、异常处理函数、矩阵生成函数、操作系统资源函数、数值计算函数、数学函数、矩阵操作函数、矩阵属性函数、图形函数、颜色函数、用户界面函数、判断函数族、mex 函数、字符串函数和类型转换函数等十几大类。其中最常用是的矩阵基础类函数和字符串函数。

MATcom 矩阵基础类数据类型为 Mm，与 C++程序中的类一样，采用 class 定义。因而，用户可像 C++中的类一样用 Mm 来定义数据对象和使用 Mm 类数据对象，Mm 类还封装众多的 Mm 类对象操作函数和析构函数，当程序终止时，程序调用 Mm 类的析构函数释放 Mm 数

据对象所占的内存空间。下面列出了 Mm 类中常用的成员变量及操作函数。

```
class Mm {
friend class Mr; ///////////// Mr 为 matlib.h 中定义的另一个类
protected:  ////////////////// 受保护成员变量
   int ndims;     ///////////// 维数
   int dims[max_ndims]; /////////// 维数数组，存放对象每一维的大小
   int p; ///////////////////////// 指向 Mm 对象的指针
   int* pc; ////////////////////// 指向 Mm 对象中复数指针
   m_type* pr; //////////////// 指向 Mm 对象中实部指针
   m_type* pi; //////////////// 指向 Mm 对象中虚部指针
   const char* self_name; /////////////类名
   const char** fields; ///////////字段名
   int nfields; ///////////////////// 字段数
   int classid;  ////////////////// 类 ID 值
.............
public: ////////////////////////////// 公有函数
   M_types flags; /////////////// 数据符号
   DLLI  Mm(); ///////////////////////////无参数构造函数，用于类对象声明
/////////////////////////////////////////////带参数的构造函数
   DLLI  Mm(int isc, int iss, int nonzeros, int nrows, int ncols, mt_matrix_types new_type=mt_double
      _matrix);
其中：isc—是否为复数矩阵；iss－是否为字符串矩阵；nonzeros－是否为非数值矩阵；
      nrows－矩阵行数；     ncols－矩阵列数；         mt_double_matrix－双精度矩阵
   DLLI  Mm(int isc, int iss, int nonzeros, int new_ndims, const int new_dims[max_ndims], mt_matrix
      _types new_type=mt_double_matrix);
   DLLI  Mm(i_o_t, const char* mname, int isglobal);
   DLLI  Mm(const char* mname, int m, int n); ///////////////
   DLLI  Mm(int aisc, cMm x, cMm y, cMm dim1, op_t op, int do_dim, Mm& minmax_idx);
   DLLI  Mm(m_type src);
   DLLI  Mm(cMm src);
   DLLI  Mm(cMm src, const char* mname);
   DLLI  Mm(const Mc& src);
   DLLI ~Mm(); ///////////////////////// 析构函数，程序终止时调用析构函数进行内存释放

   Mm RDLLI operator =(cMm src);///////赋值操作符重载

   inline int DLLI rows() const { return dims[0]; } /////////返回矩阵的行数
   inline int DLLI cols() const { return dims[1]; }   ////////返回矩阵的列数
   int DLLI size() const; ///////返回 Mm 对象的元素个数大小
   int DLLI size(int dim) const; /////返回 Mm 对象每一维的元素个数
   int DLLI vectordim() const; /////返回向量的维数

   const char PDLLI getuame() const { return self_name; }///获取类名
   void DLLI setname(const char* new_name);//////////////////设置新的类名

   int DLLI isstr() const { return (flags.str!=0); }////判断是否为字符矩阵
   inline int DLLI issparse() const { return (flags.sparse!=0); }////判断是否为稀疏矩阵
   inline int DLLI isglobal() const { return (flags.global!=0); }////判断是否为全局矩阵
   inline int DLLI isstruct() const { return (fields!=NULL); }////判断是否为结构体矩阵
   inline int DLLI isc() const { return (pi!=NULL); }////判断是否为复数矩阵
```

```
    inline int DLLI islogical() const { return (flags.logical!=0); }////判断是否为逻辑变量矩阵

    inline int RDLLI getndims() const { return (int&)ndims; }/////获取维数数组指针

    void DLLI setstr(int newd);////////设置字符串
    void DLLI setsparse(int sp);/////设置稀疏矩阵
    void DLLI setlogical(int newd);/////设置逻辑变量
    inline M_types RDLLI getflags() { return flags; }///获取数据符号

    inline m_type PDLLI getpr(m_type*) const { return (m_type*)pr; }///获取实部指针
    inline m_type PDLLI getpi(m_type*) const { return (m_type*)pi; }///获取虚部指针
    m_type PDLLI addr() const; ////////// 获取指向 Mm 对象的实部指针起始地址
    m_type PDLLI addr(int i0) const; ///获取指向 Mm 对象的虚部指针起始地址
    m_type PDLLI addr(int i0,int i1) const; //获取指向 Mm 对象(i0,i1) 实部地址
    m_type PDLLI addi() const; ////////// 获取指向 Mm 对象的虚部起始地址
    m_type PDLLI addi(int i0) const; /// 获取指向 Mm 对象中第 i0 个元素的虚部地址
    m_type PDLLI addi(int i0,int i1) const; /获取指向 Mm 对象(i0,i1) 虚部地址

    int DLLI getreal(int force=0) const;////////获取实部
    int DLLI getcomplex();//////////////////////////获取复数

    int DLLI findfield(const char* field, int err) const;////查找字符串
    const char PDLLI getfield(int i) const;////////////////// 按索引号获取字符串指针
    int DLLI addfield(const char* field,int quick);///////// 按索引号添加字符串
    int DLLI getclassid() const { return classid; }/////////获取类 ID
    void DLLI setclassid(int new_classid) { classid=new_classid; }///设置新的 ID
......
}; // M
```

在上述 Mm 类定义基础上，举例说明 Mm 类对象的定义、赋值和运算等操作。下面程序清单为 VC++的主要程序片段。

（1）使用 Mm 类，必须包含头文件：

```
#include "matlib.h"
//////////////////////////////
```

（2）Mm 对象定义、赋值、运算等举例

```
////////////Mm 对象定义、赋值、运算等举例
int i,j,m,n;
double **A1,**B1,**C1,**D1;
Mm X1,X2,X3,X4,X5; ////用无参数的构造函数定义 Mm 类对象
FILE* fp;
if((fp=fopen("Mm.txt","w"))==NULL){MessageBox("Can't open R_R Input File");exit(0);}
m=n=5;
A1=new double*[m];for(i=0;i<m;i++)A1[i]=new double[n];//C 动态数组由用户创建并分配地址
B1=new double*[m];for(i=0;i<m;i++)B1[i]=new double[n];
C1=new double*[m];for(i=0;i<m;i++)C1[i]=new double[n];
D1=new double*[m];for(i=0;i<m;i++)D1[i]=new double[n];

X1=magic(n); //////////生成一个 3 阶魔方矩阵，由 MATlib 矩阵直接赋值，赋值时隐含矩阵方式初始化
X2=X1;        /////Mm 类实现了操作符"="重载，赋值时隐含矩阵方式初始化
```

```
//Mm ii,jj;
X3=X1+X2; /////Mm 类实现了操作符"+"重载，赋值时隐含矩阵方式初始化
X4=X1*X2; /////Mm 类实现了操作符"*"重载(矩阵乘法)，赋值时隐含矩阵方式初始化
for(i=0;i<n;i++)for(j=0;j<n;j++)
{A1[i][j]=*X1.addr(i+1,j+1);// .addr(i+1,j+1)获取(i+1,j+1)元素实部地址，并将数据赋给 A1[i][j]
B1[i][j]=*X3.addr(i+1,j+1);// 组下标从 0 开始，而 Mm 矩阵下标从 1 开始，故(i+1,j+1)与[i][j]对应
C1[i][j]=X4.r(i+1,j+1);      // .r(i+1,j+1)直接读取 Mm 对象的实部(i+1,j+1)中数据赋给 C1[i][j]
D1[i][j]=(*X1.addr(i+1,j+1))*(*X2.addr(i+1,j+1));
*X5.addr(i+1,j+1)=D1[i][j];//X5 虽由 Mm 定义对象，但未经矩阵方式初化而直接引用 Mm 对象中元素，
                //////////////////这种方式极易出错，建议不使用未经矩阵方式初始化的 Mm 对象
}
//////////////////输出操作
fprintf(fp,"\n Output X1 by *X1.addr(i+1,j+1):\n");
for(i=0;i<n;i++){fprintf(fp,"\n");for(j=0;j<n;j++)fprintf(fp,"  %8.3lf ",*X1.addr(i+1,j+1));}
fprintf(fp,"\n Output X2 by X2.r(i+1,j+1) :\n");
for(i=0;i<n;i++){fprintf(fp,"\n");for(j=0;j<n;j++)fprintf(fp,"  %8.3lf ",X2.r(i+1,j+1));}
fprintf(fp,"\n Output X1+X2 by B1[i][j]:\n");
for(i=0;i<n;i++){fprintf(fp,"\n");for(j=0;j<n;j++)fprintf(fp,"  %8.3lf ",B1[i][j]);}
fprintf(fp,"\n Output X1*X2 by C1[i][j]:\n");
for(i=0;i<n;i++){fprintf(fp,"\n");for(j=0;j<n;j++)fprintf(fp,"  %8.3lf ",C1[i][j]);}
//////////////////////////////////////////////
fprintf(fp,"\n Output X1.*X2 by D1[i][j] :\n");
for(i=0;i<n;i++){fprintf(fp,"\n");for(j=0;j<n;j++)fprintf(fp,"  %8.3lf ",D1[i][j]);}
fprintf(fp,"\n Output X1.*X2 by D1[i][j] :\n");
for(i=0;i<n;i++){fprintf(fp,"\n");for(j=0;j<n;j++)fprintf(fp,"  %8.3lf ",*X5.addr(i+1,j+1));}
//////////D1[i][j]输出数据应与*X5.addr(i+1,j+1)输出数据相同，但程序运行结果不同，表明，直接
//引用未经矩阵方式初始化的 Mm 对象元素，易导数错误结果,*X5.addr(i+1,j+1)输出结果错误

Mm Y1=ones(n); ///定义对象时隐含矩阵方式初始化
Mm Y2(" ",m,n); ///或采用带参数 m,n 的构造函数定义对象
for(i=0;i<n;i++)for(j=0;j<n;j++)
{
 Y1.r(i+1,j+1)=A1[i][j]; ////////可引用 Mm 对象中的元素
 *Y2.addr(i+1,j+1)=B1[i][j];////////可引用 Mm 对象中的元素
}
fprintf(fp,"\n Output Y1 by Y1.r(i+1,j+1) :\n");
for(i=0;i<n;i++){fprintf(fp,"\n");for(j=0;j<n;j++)fprintf(fp,"  %8.3lf ",Y1.r(i+1,j+1));}
fprintf(fp,"\n Output Y2 by *Y2.addr(i+1,j+1) :\n");
for(i=0;i<n;i++){fprintf(fp,"\n");for(j=0;j<n;j++)fprintf(fp,"  %8.3lf ",*Y2.addr(i+1,j+1));}
////////////////////   Y1.r(i+1,j+1)和*Y2.addr(i+1,j+1)输出正确

//////////////////// 使用宏 BR()对 Mm 对象赋初值举例
Mm Z;
Z=(BR(10),1,3,5,7,semi,6,7,8,9,10,semi,12,17,18,19,20);//赋值时隐含用矩阵初始化 Z 对象
///BR()宏表示 Z 矩阵开始，BR(10)表示 Z 矩阵第一个元素值为 10，以后依次将数据赋与 Z,
///semi 宏表示分号
fprintf(fp,"\n Output Z by *Z.addr(i+1,j+1) :\n");
for(i=0;i<3;i++){fprintf(fp,"\n");for(j=0;j<5;j++)fprintf(fp,"  %8.3lf ",*Z.addr(i+1,j+1));}

//////////////////// 使用宏 M_VECTOR 对 Mm 对象赋初值举例
```

```
double E_Vector[6]={1,2,3,4,5,6};
Mm Z1; M_VECTOR(Z1,E_Vector); // 使用宏 M_VECTOR 对 Mm 对象赋初值
fprintf(fp,"\n Output Z1 by *Z.addr(1,j+1) :\n");
for(j=0;j<6;j++)fprintf(fp," %8.3lf ",*Z1.addr(1,j+1));
for(i=0;i<m;i++)delete A1[i];delete A1; //由用户释放内存空间，而 Mm 对象则自动调用
for(i=0;i<m;i++)delete B1[i];delete B1; //析构函数来释放所分配的空间
for(i=0;i<m;i++)delete C1[i];delete C1;
for(i=0;i<m;i++)delete D1[i];delete D1;
/////////////////字符串操作举例
Mm Z2=("",m,n);
CString str[5];
str[0]="MATcom is based on the Mm";
str[1]="Mm is defined in matlib.h";
str[2]="Read it carefully";
str[3]="Learn the data exhange mechanism";
str[4]="You'll sigh the MATcom's programming is a piece of cake with emotion";
for(i=0;i<m;i++)Z2.addfield(str[i],i+1);///使用.addfield 将字符串添加到 Mm 对象中
fprintf(fp,"\n Output Z2 by Z2.getfield() :\n");
for(i=0;i<m;i++)fprintf(fp,"\n  %s \n",Z2.getfield(i));///////按索引号获取字符串指针

fclose(fp); ////////////关闭文件
//////////////////////////////////////////结束//////////////////////////////////////////
```

程序运行后的输出结果如下：

```
Output X1 by *X1.addr(i+1,j+1):
   17.000    24.000     1.000     8.000    15.000
   23.000     5.000     7.000    14.000    16.000
    4.000     6.000    13.000    20.000    22.000
   10.000    12.000    19.000    21.000     3.000
   11.000    18.000    25.000     2.000     9.000
 Output X2 by X2.r(i+1,j+1) :
   17.000    24.000     1.000     8.000    15.000
   23.000     5.000     7.000    14.000    16.000
    4.000     6.000    13.000    20.000    22.000
   10.000    12.000    19.000    21.000     3.000
   11.000    18.000    25.000     2.000     9.000
 Output X1+X2 by B1[i][j]:
   34.000    48.000     2.000    16.000    30.000
   46.000    10.000    14.000    28.000    32.000
    8.000    12.000    26.000    40.000    44.000
   20.000    24.000    38.000    42.000     6.000
   22.000    36.000    50.000     4.000    18.000
 Output X1*X2 by C1[i][j]:
 1090.000   900.000   725.000   690.000   820.000
  850.000  1075.000   815.000   720.000   765.000
  700.000   840.000  1145.000   840.000   700.000
  765.000   720.000   815.000  1075.000   850.000
  820.000   690.000   725.000   900.000  1090.000
 Output X1.*X2 by D1[i][j] :
  289.000   576.000     1.000    64.000   225.000
```

```
529.000    25.000    49.000   196.000   256.000
 16.000    36.000   169.000   400.000   484.000
100.000   144.000   361.000   441.000     9.000
121.000   324.000   625.000     4.000    81.000
Output X1.*X2 by D1[i][j] :
225.000   225.000   225.000   225.000   225.000
256.000   256.000   256.000   256.000   256.000
484.000   484.000   484.000   484.000   484.000
  9.000     9.000     9.000     9.000     9.000
 81.000    81.000    81.000    81.000    81.000
Output Y1 by Y1.r(i+1,j+1) :
 17.000    24.000     1.000     8.000    15.000
 23.000     5.000     7.000    14.000    16.000
  4.000     6.000    13.000    20.000    22.000
 10.000    12.000    19.000    21.000     3.000
 11.000    18.000    25.000     2.000     9.000
Output Y2 by *Y2.addr(i+1,j+1) :
 34.000    48.000     2.000    16.000    30.000
 46.000    10.000    14.000    28.000    32.000
  8.000    12.000    26.000    40.000    44.000
 20.000    24.000    38.000    42.000     6.000
 22.000    36.000    50.000     4.000    18.000
Output Z by *Z.addr(i+1,j+1) :
 10.000     1.000     3.000     5.000     7.000
  6.000     7.000     8.000     9.000    10.000
 12.000    17.000    18.000    19.000    20.000
Output Z1 by *Z.addr(1,j+1) :
  1.000     2.000     3.000     4.000     5.000     6.000
Output Z2 by Z2.getfield :
MATcom is based on the Mm
Mm is defined in matlib.h
Reed it carefully
Learn the data exhange mechanism
You'll sigh the MATcom's programming is a piece of cake with emotion
```

第 10 章　MATLAB 与 Excel 接口

前面曾经介绍了 MATLAB 的 Excel 生成器，利用它可以生成 DLL 组件和 VBA 代码。在 Excel 中，可以把 VBA 代码保存为 Excel 插件。利用 DDE 和自动化技术也能实现它们之间的链接。此外，MATLAB 还提供了一个 Excel Link 插件，利用它，可以直接在 Excel 环境下完成与 MATLAB 的数据传输，并运行 MATLAB 命令。所以，MATLAB 与 Excel 的接口是目前 MATLAB 与外部各种程序的接口中手段最多、最完备的。

10.1　自动化链接

10.1.1　MATLAB 作为自动化客户端

利用 MATLAB 的 actxserver 函数，可以将 Excel 设置为服务器端应用程序。

```
h=actxserver('excel.application')
h=
    COM.excel.application
```

用 get 函数获取一个 Excel 应用的工作簿接口。

```
w=get(h, 'workbooks')
w=
    Interface.excel.application.workbooks
```

用 get 函数获取对象所有属性值

```
get(h)
ans =
    Application: [1x1 Interface.excel.application.Application]
          Creator: 'xlCreatorCode'
           Parent: [1x1 Interface.excel.application.Parent]
     ActiveCell: []
    ActiveChart: [1x50 char]
  ActivePrinter: '\\PRINTERS\Kunkel on Ne01:'
    ActiveSheet: []
   ActiveWindow: []
          AddIns: [1x1 Interface.excel.application.AddIns]
             :
             :
```

下面介绍一个 MATLAB 自己附带的例子。本例中，把 MATLAB 作为自动化客户端，把 Excel 作为服务器端。

```
% 首先，打开一个 Excel 服务器
e = actxserver('excel.application');

% 插入一个新的工作簿
eWorkbook = e.Workbooks.Add;
```

```
e.Visible = 1;

% 激活第 1 个表
eSheets = e.ActiveWorkbook.Sheets;

eSheet1 = eSheets.get('Item', 1);
eSheet1.Activate;

% 把 MATLAB 数组放到 Excel
A = [1 2; 3 4];
eActivesheetRange = e.Activesheet.get('Range', 'A1:B2');
eActivesheetRange.Value = A;

% 返回一个范围，它是一个单元数组，因为单元范围可以包含不同类型的数据
eRange = e.Activesheet.get('Range', 'A1:B2');
B = eRange.Value;

% 转换为 double 矩阵，单元数组必须只包含标量
B = reshape([B{:}], size(B));

% 保存工作簿
eWorkbook.SaveAs('myfile.xls');

% 退出 Excel 并删除服务器
e.Quit;
e.delete;
```

在命令窗口中输入以上命令以后，启动 Excel 并在工作簿中输入一个 MATLAB 数组的值，然后保存为“myfile.xls”，如图 10-1 所示。

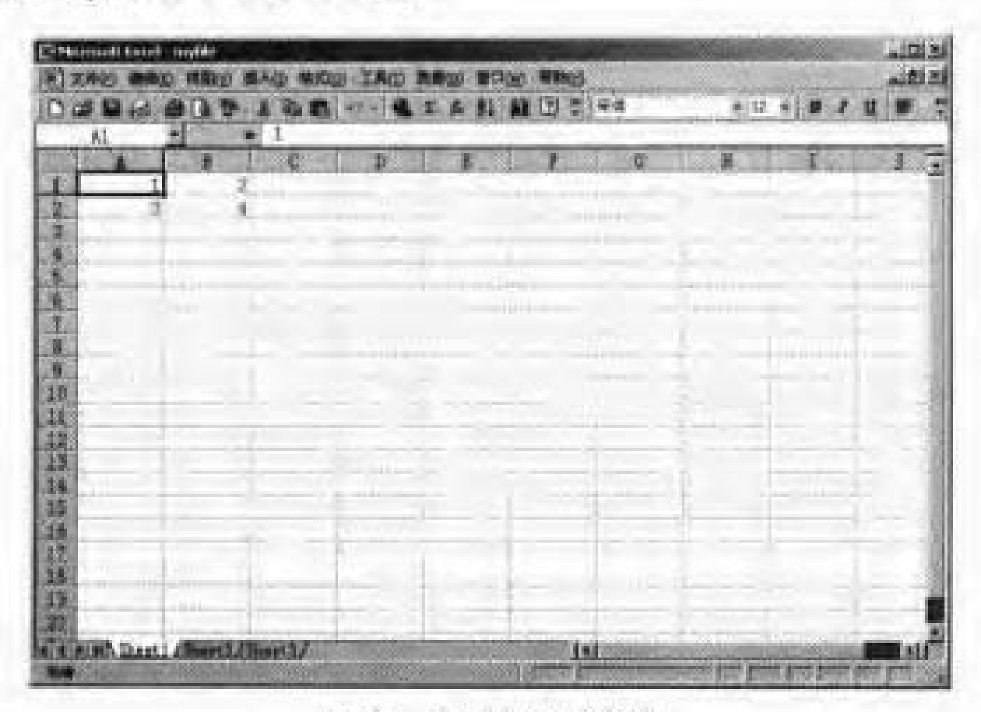


图 10-1　Excel 作为服务器端

10.1.2 MATLAB 作为自动化服务器端

把 MATLAB 作为自动化服务器端时，在 Excel 的 Visual Basic 编辑器中编程，利用 CreateObject 函数或 GetObject 函数，结合 MATLAB 的 ProgID 值，可以在 Excel 中直接启动 MATLAB，并使用 MATLAB 提供的自动化方法和属性进行编程。

Excel 的 Visual Basic 编辑器如图 10-2 所示。

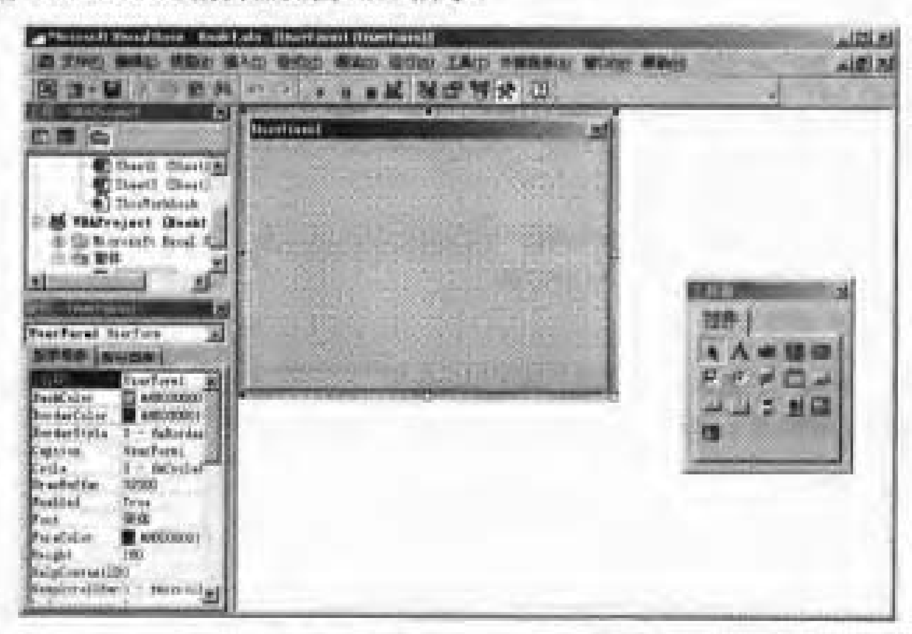


图 10-2 Excel 的 Visual Basic 编辑器

10.2 Excel Link 插件

10.2.1 概述

Excel Link 是一个软件插件，它将 Excel 和 MATLAB 在微软视窗环境下进行集成。通过链接 Excel 和 MATLAB，可以从 Excel 工作表和宏编程工具中获得 MATLAB 的数值计算和图形绘制功能，能够在两个环境之间交换数据。

Excel Link 在 Excel 工作空间和 MATLAB 工作空间之间进行通信。它把 Excel 作为 MATLAB 的前端，在 Excel 的工作表或宏中使用 Excel Link 函数，而不必离开 Excel 环境。Excel Link 用少量的函数实现链接管理和数据操作，以简便取胜。

Excel Link 的运行机制如图 10-3 所示。

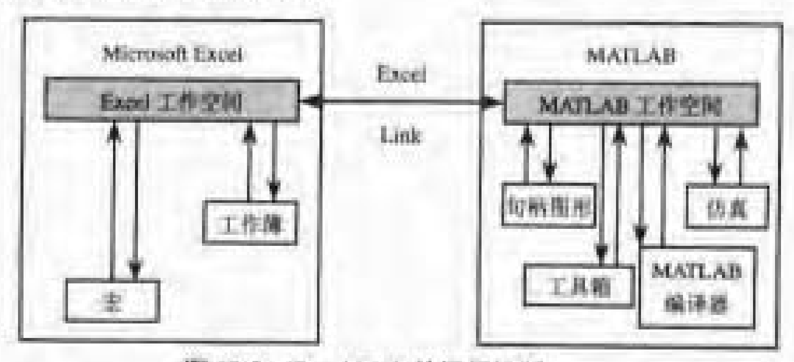


图 10-3 Excel Link 的运行机制

10.2.2 安装和操作 Excel Link 插件

1. 系统需求

Excel Link 占用近 202KB 的磁盘空间。操作系统可以是 Windows 98，Windows NT 4.0 或 Windows 2000。Excel Link 还需要 Excel 97 或 Excel 2000，和 5.1 以上版本的 MATLAB。

为了使 MATLAB 的图形图像有更好的视觉效果，将显示器设置为 256 种颜色以上。

在安装 MATLAB 的过程中，选择 Excel Link 核选框，就可以安装 Excel Link。

2. 在 Excel 中注册 Excel Link

安装 Excel Link 以后，准备注册 Excel。需要进行以下几个步骤。

（1）启动 Excel。

（2）在“工具”菜单中选择“加载宏...”选项，打开“加载宏”对话框，如图 10-4 所示。在对话框中单击“浏览...”按钮。

（3）在<matlab>/toolbox/exlink 目录中找到和选择 Excel 插件 excllink.xla，单击“确定”按钮。



图 10-4 “加载宏”对话框

（4）回到“加载宏”窗口，可以看到在“当前加载宏”列表框的顶部添加了“Excel Link 2.0 for use with MATLAB”宏，而且该选项被选中。单击“确定”按钮，现在安装了 Excel Link 插件。

（5）显示“MATLAB Command Window”窗口，如图 10-5 所示。

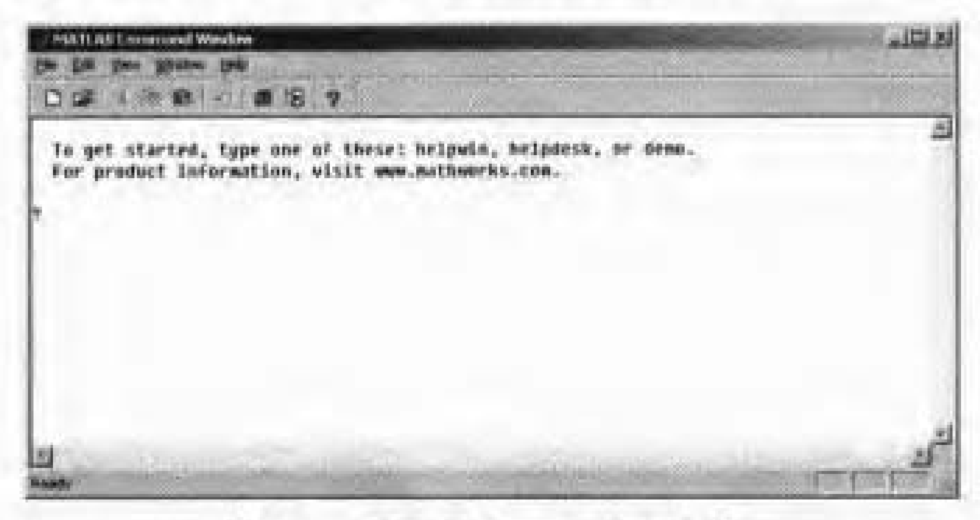


图 10-5 “MATLAB Command Window”窗口

（6）稍等片刻，Excel Link 工具条在 Excel 工作表中出现，如图 10-6 所示。添加的 Excel Link 工具条中包括 3 个工具按钮，即“putmatrix”、“getmatrix”和“evalstring”，它们分别实现把数据传给 MATLAB、从 MATLAB 提取数据和执行 MATLAB 命令。

现在 Excel Link 可以用了。

注意：<matlab>指 MATLAB 安装的根目录，这时 MATLAB 桌面不会自动启动，如果试图运行桌面，则在命令窗口中输入“desktop”。

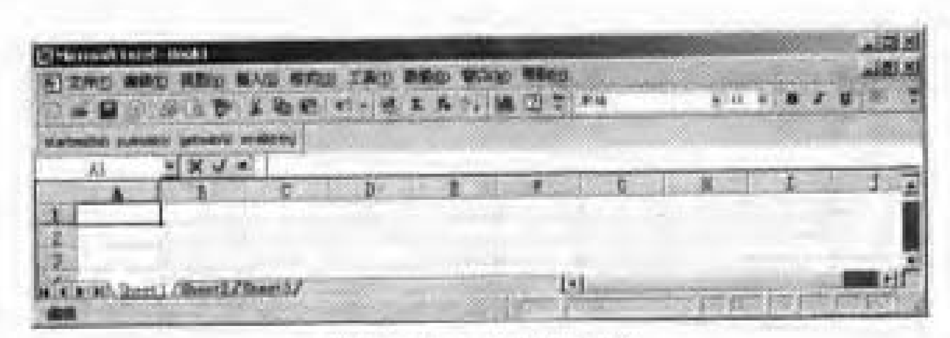


图 10-6　Excel Link 工具条

3．启动 Excel Link

可以自动启动 Excel Link，也可以用手工启动它，下面分别进行介绍。

（1）自动启动：按照前面的提示安装和注册以后，启动 Excel 时，Excel Link 和 MATLAB 会自动启动。如果不希望启动 Excel 时自动启动 Excel 和 MATLAB，在工作表单元中输入“=MLAutoStart("no")”，该函数会改变初始化文件，这样，Excel Link 和 MATLAB 不再自动启动。

（2）手工启动：从 Excel 中手工启动 Excel Link 和 MATLAB，可在“工具”菜单中选择“宏”，然后在它的次级菜单中单击“宏...”选项，打开“宏”对话框，如图 10-7 所示。在“宏名”文本框中输入“matlabinit”并单击“执行”按钮。等一会，就可看到“MATLAB Command Window”按钮显示在任务条上。

4．链接已经存在的 MATLAB

要将一个新的 Excel 进程与已经存在的 MATLAB 进程链接起来，必须用/automation 命令行选项启动 MATLAB，该选项把 MATLAB 作为一个自动化服务器，并使命令窗口最小化。

按照下面的步骤在命令行中添加/automation 选项：

① 右击 MATLAB 的快捷图标；

② 选择“属性”选项；

③ 单击“快捷方式”选项卡；

④ 在“目标”文本框中添加字符串 /automation。记住，在 matlab.exe 和/automation 之间留一个空格。

完成以上步骤后的窗口如图 10-8 所示。

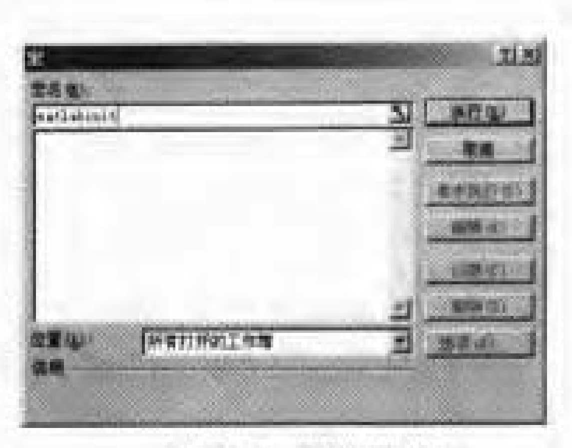


图 10-7　“宏”对话框

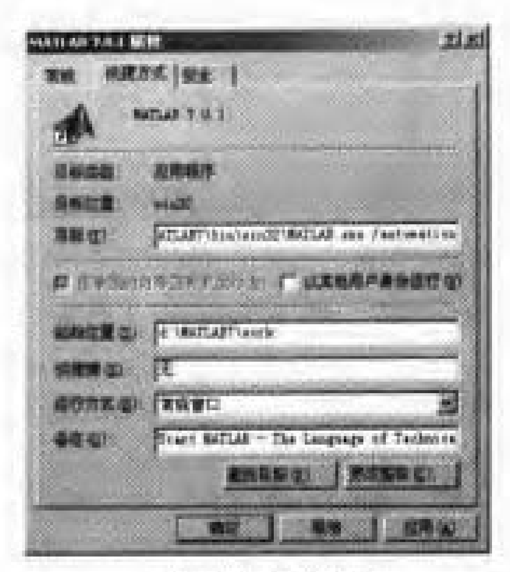


图 10-8　链接设置

5. 终止 Excel Link 的运行

退出 Excel，也就同时退出了 Excel Link 和 MATLAB。要想停止运行 MATLAB 和 Excel Link，但让 Excel 继续运行，应在 Excel 工作表单元中输入“=MLClose()”。可以用 MLOpen 或 matlabinit 手工重启 Excel Link 和 MATLAB。如果直接在 MATLAB 命令窗口中停止运行 MATLAB，并让 Excel 继续运行，应在 Excel 工作表单元中输入“=MLClose()”。可以用 MLOpen 或 matlabinit 手工重启 Excel Link 和 MATLAB。

通过 Excel Link 这个中介，Excel 成为 MATLAB 的一个易于使用的数据存储和应用开发前端，它是一个威力强大的计算和图形处理器。

10.2.3 Excel Link 的函数

Excel Link 提供了一些管理链接和操作数据的函数，利用它们，可以不必离开 Excel 环境，像调用工作表单元公式那样调用它们或者在宏中使用它们。

1. 链接管理函数

Excel Link 提供了 4 个链接管理函数来初始化、启动、终止 Excel Link 和 MATLAB。

- matlabinit　　初始化 Excel Link 并启动 MATLAB 进程。
- MLAutoStart　　自动启动 MATLAB 进程。
- MLClose　　终止 MATLAB 进程。
- MLOpen　　启动 MATLAB 进程。

可作为一个工作表单元公式或在宏中调用任何除了 matlabinit 以外的链接管理函数。在 Excel 的“工具”菜单中单击“宏”选项或在宏过程中调用 matlabinit 函数。

使用 MLAutoStart 自动启动。如果根据默认提示安装和注册 Excel Link，则 Excel Link 和 MATLAB 在每次启动 Excel 时将被自动启动。如果选择手工启动，则使用 matlabinit 初始化 Excel Link 并启动 MATLAB。

使用 MLClose 在不停止运行 Excel 的情况下停止运行 MATLAB，并使用 MLOpen 或 matlabinit 重启 MATLAB。

2. 数据管理函数

Excel Link 提供了 9 个数据管理函数在 Excel 和 MATLAB 之间复制数据并从 Excel 运行 MATLAB 命令。

- matlabfcn　　对于给定的 Excel 数据运行 MATLAB 命令。
- matlabsub　　对于给定的 Excel 数据运行 MATLAB 命令并指定输出位置。
- MLAppendMatrix　　将 Excel 工作表中的数据创建或添加到 MATLAB 矩阵。
- MLDeleteMatrix　　删除 MATLAB 矩阵。
- MLEvalString　　运行 MATLAB 命令。
- MLGetMatrix　　把 MATLAB 矩阵的内容写到 Excel 工作表。
- MLOetVar　　把 MATLAB 矩阵的内容写到 Excel VBA 变量中。
- MLPutMatrix　　用 Excel 工作表中的数据创建或覆盖 MATLAB 矩阵。
- MLPutVar　　用 Excel VBA 变量的数据创建或覆盖 MATLAB 矩阵。

除了 MLGetVar 和 MLPutVar 以外，其他函数都可以作为工作表单元公式使用或在宏中被调用。MLGetVar 和 MLPutVar 函数，只能在宏中调用，而不能作为工作表单元公式使用。

10.2.4 技巧和提示

当 Excel 函数返回一个值时，Exce Link 完成一次调用。在使用 Excel Link 时请注意下面一些问题。

1．语法

（1）Excel Link 的函数名对大小写不敏感，即 MLPutMatrix 和 mlputmatrix 是相同的。

（2）MATLAB 的函数名与变量名对大小写敏感，即 BONDS, Bonds 和 bonds 是 3 个不同的 MATLAB 变量。标准的 MATLAB 函数名是区分大小写的，如 plot(f)。

（3）工作表公式用 + 或 = 开始，如：

=mlputmatrix("a",C10)

（4）在工作表公式中，把函数变量用小括号括起来，在宏中，在函数名和第 1 个变量之间留一个空格。

（5）在大部分 Excel Link 函数中可以直接或间接指定一个变量名称变量。

（6）直接指定一个变量名，用双引号把它括起来，如 MLDeleteMatrix("Bonds")。

（7）用一个没有引号的变量名称参数进行间接引用，函数评价变量的内容，以获取变量名。变量必须是一个工作表单元地址或范围名称。

（8）一个数据位置变量必须是一个工作表单元地址或范围名称。数据位置变量用不同引号括起来（MLGetMatrix 除外，它有一个惟一的变量提示）。

（9）数据位置变量可以包括一个工作表数目，如 Sheet3！B1：C7 或 Sheet2！OUTPUT。

2．工作表

（1）Excel Link 函数作为工作表公式成功运行以后，单元包含 0 值。执行函数时，单元会继续显示输入的公式。

（2）假设在 Excel 的“工具”菜单中单击“选项...”选项，然后在打开的“选项”对话框中选择“编辑”选项卡，在该选项卡中选择“按 Enter 键后移动”核选框。激活的单元在操作完成时改变，为长时间的操作提供一个有用的确认。

（3）建议用自动化计算模式使用 Excel Link。如果用手工计算模式使用 MLGetMatrix，则在单元中输入函数，并按下 F9 键运行它。在这种情况下按下 F9 键还可以重新运行其他工作表函数并生成不可预测的结果。

（4）要想在工作表中重新计算 Excel Link 函数，单击 F2 键，然后回车，可以重新运行每个函数。

（5）单击 F9 键，工作表只影响 Excel 函数（它返回一个值），不对 Excel Link 函数进行操作。

（6）要使 Excel Link 函数的重新计算能自动进行，将它添加到值发生改变的单元中，如：

=MLPutMatrix("bonds",D1:O26)+C1

（7）单元 C1 中的值发生改变时，Excel 重新运行 MLPutMatrix 函数。注意，不管怎么样，不要产生死循环。

（8）Excel Link 函数希望有 A1 类型单元表的单元引用。在 Excel 中，在“工具”菜单中单击“选项...”选项，打开“选项”对话框，选择“常规”选项卡，然后选择“A1 单元引用样式”核选框。

（9）如果在 MLGetMatrix 中使用单元地址，并在以后插入或删除行或列，或把函数移动或复制到另一个单元，则编辑变量以更正地址。Excel Link 不自动在 MLGetMatrix 中调整单元地址。

（10）在工作表单元中直接输入 Excel Link 函数，不要使用 Excel 函数大师（它会生成不可预料的结果）。

3．宏

（1）用 Excel Link 函数创建宏，必须首先注册 Excel，以便引用源于 Excel Link 插件的函数。从 Visual Basic 环境中拉下“插入”菜单并选择“模块”。打开“模块”页时，打开“工具”菜单并在“引用”对话框中选择“引用…”选项。选择“excllink.xla”并单击“确定”按钮。用“浏览…”按钮查找“excllink.xla”文件。

（2）如果在宏过程中使用 MLGetMatrix，在命令行中，MLGetMatrix 后面输入 MATLABRequest。MATLABRequest 初始化内部 Excel Link 变量并使 MLGetMatrix 可用。如：

```
Sub Get_RangeA()
    MLGetMatrix “A” , “RangeA”
    MATLABRequest
End Sub
```

（3）除非宏函数从过程中调用，不要在宏函数中包括“MATLABRequest”。

4．数据类型

Excel Link 只控制 MATLAB 二维数值数组。一维字符数组（字符串）和只包含字符串的二维单元数组。它不能用于 MATLAB 多维数组、结构或包含字符以外数据的单元数组。

5．日期

默认的 Excel 日期从 1900 年 1 月 1 日开始，而 MATLAB 的日期从 0000 年 1 月 1 日开始。这样，1996 年 5 月 1 日在 Excel 中是 35200，在 MATLAB 中是 729160，相差 693960。如果在 MATLAB 计算中使用日期数字，将 Excel 日期换为 MATLAB 日期时要加上常量 693960。如果使用的是可选的 Excel 1904 日期系统，则常量是 695422。

6．保存工作表

（1）打开包含 Excel Link 函数的 Excel 工作表时，Excel 试图从下向上、从右向左运行该函数，这样，可能会生成单元错误信息（#COMMAND!.#NONEXIST!等）。Excel 常会发生这种情况。解决这个问题，只需要简单地忽略该信息，关闭所有 MATLAB 图形窗口，并单击 F2 键，以正确的秩序每次重新运行一个单元函数，然后回车。

（2）如果保存包含 Excel Link 函数的 Excel 工作表，以后在不同的计算机环境下打开时，则 excllink.xla 插件将位于不同的位置，Excel 可能会显示一个信息框，提示是否重新建立链接。

（3）在对话框中单击“否”按钮，然后在“编辑”菜单中选择“链接…”选项。在“链接”对话框中，单击“改变链接”，在窗口中的<matlab>/toolbox/exlink 目录中找到并选择 exclinkxla，然后单击“确定”按钮。Excel 在改变链接时运行每个函数。当链接改变并且函数运行时，可以看到 MATLAB 图形窗口并听到发生错误时的蜂鸣声，忽略它们。回到“链接”窗口，单击“确定”按钮。现在工作表就正确链接到 Excel Link 插件了。或者不用“编辑”菜单中的“链接…”选项，可以编辑每个受影响的工作表单元中的链接位置，以显示 excllink.xla 的正确位置。

10.2.5 Excel Link 使用实例

【实例 1】 回归和曲线拟合。

回归和曲线拟合方法试图用合适的函数来描述变量之间的关系，即建立数字模型。MATLAB 提供了很多功能强大且易于使用的矩阵操作符和函数来简化这项工作。

本例运行同一实例的工作表版本和宏版本，它使用 Excel 工作表组织和显示数据。Excel Link 函数将数据复制到 MATLAB 并运行 MATLAB 的计算和绘图函数。宏版本还把输出数据返回到 Excel 工作表中。

本实例 ExliSamp.xls 位于<matlab>\toolbox\exlink 目录下，其中，<matlab>为 MATLAB 的安装目录。

1．工作表版本

在 ExliSamp.xls 中单击“Sheet1”选项卡，尝试运行本例的工作表版本，如图 10-9 所示。

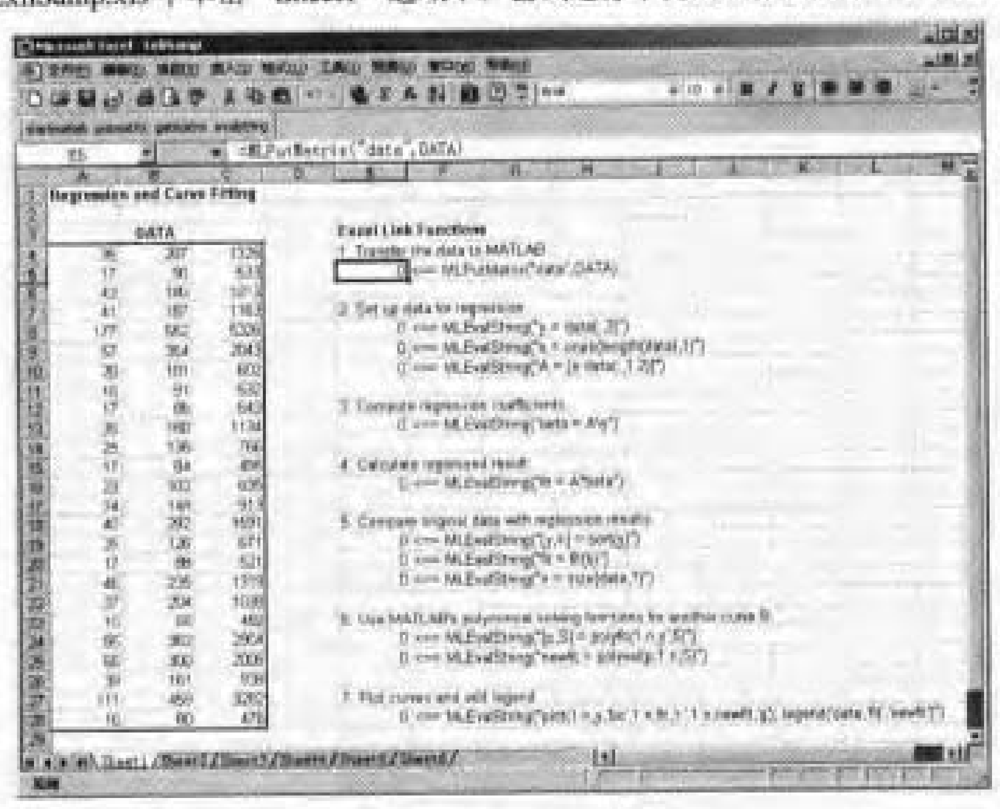


图 10-9　Exlisamp.xls 文件

工作表包括一个命令范围，一个名为 DATA 的范围 A4:C28，它包含实例的数据集。按照下面的步骤进行操作。

（1）激活单元 E5，单击 F2，然后回车，运行将示例数据复制到 MATLAB 的 Excel Link 函数。数据集包括 3 个变量的 25 个观测值，观测量显示，变量之间存在较强的线性关系。

（2）移至单元 E8，单击 F2，回车，对 E9 和 E10 重复上面的操作，这些 Excel Link 函数让 MATLAB 以第 1 列和第 2 列的数据对应的变量为自变量，以第 3 列数据对应的变量为因变量进行回归，将创建一个包含第 3 列数据的一维矢量 Y 和一个新的 3 列矩阵 A，A 中第 1 列全是 1，后面跟着其他数据。

（3）运行 E13 中的函数，本函数用 MATLAB 的“\”运算符计算回归系数，以便求解线

性方程 A*beta=y 这样一个系统。

（4）运行 E16 中的函数。MATLAB 的矩阵—矢量乘法生成的回归结果(fit)。

（5）运行 E19, E20 和 E21 中的函数。这些函数将原始数据与 fit 进行比较，将数据按升序排列并对 fit 采用相同的 permutation，并创建一个表示观测量数目的标量。

（6）运行 E24 和 E25 中的函数。它们用于拟合一个多项式方程，polyfit 函数生成一个五阶多项式 polyval，然后计算每个数据点上多项式的结果，确定拟合的精度(newfit)。

（7）运行 E28 中的函数。MATLAB 的 plot 函数用图形显示原始数据（带圈），回归结果（虚红线）和多项式结果（实绿线），并添加图例，如图 10-10 所示。

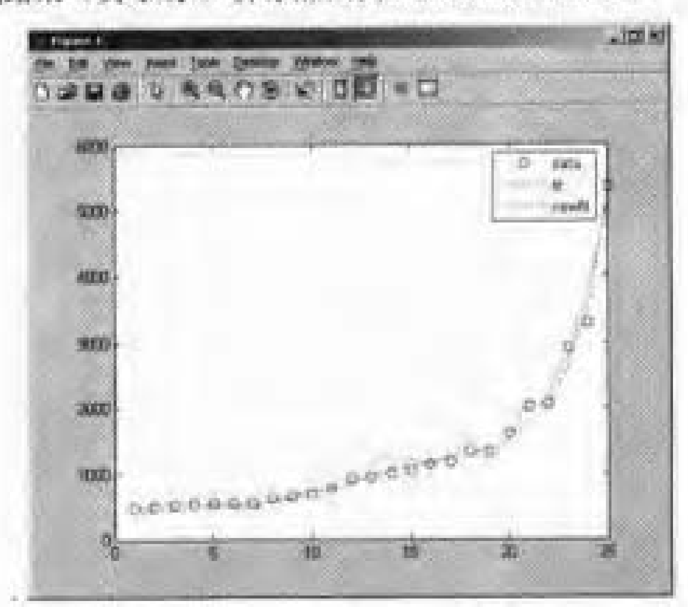


图 10-10　回归和曲线拟合结果

2. 宏版本

要试用本例的宏数据表版本，在 ExliSamp.xls 中单击 Sheet2 选项卡。

激活单元 A4，但不要运行。

单元 A4 调用宏 CurveFit，它可以从 Visual Basic 环境中进行测试，如图 10-11 所示。



图 10-11　宏 CurveFit

打开本模块时，在“工具”下拉菜单中选择“引用...”选项，打开“引用”对话框，在该对话框中选择“exllink.xla”核选框。

回到 Sheet2 中的 A4 单元，按下 F2 键，然后回车，运行 CurveFit 宏。宏运行工作表版本中第（1）步到第（7）步相同的函数（顺序稍有不同），包括图形绘制。另外，它还将经过排序的原始数据 Y、对应的回归数据 fit 和多项数据 newfit 复制到工作表中（CutveFit 宏中最后 3 个 MLGetMatrix 函数将数据复制到 Excel 工作表），如图 10-12 所示。

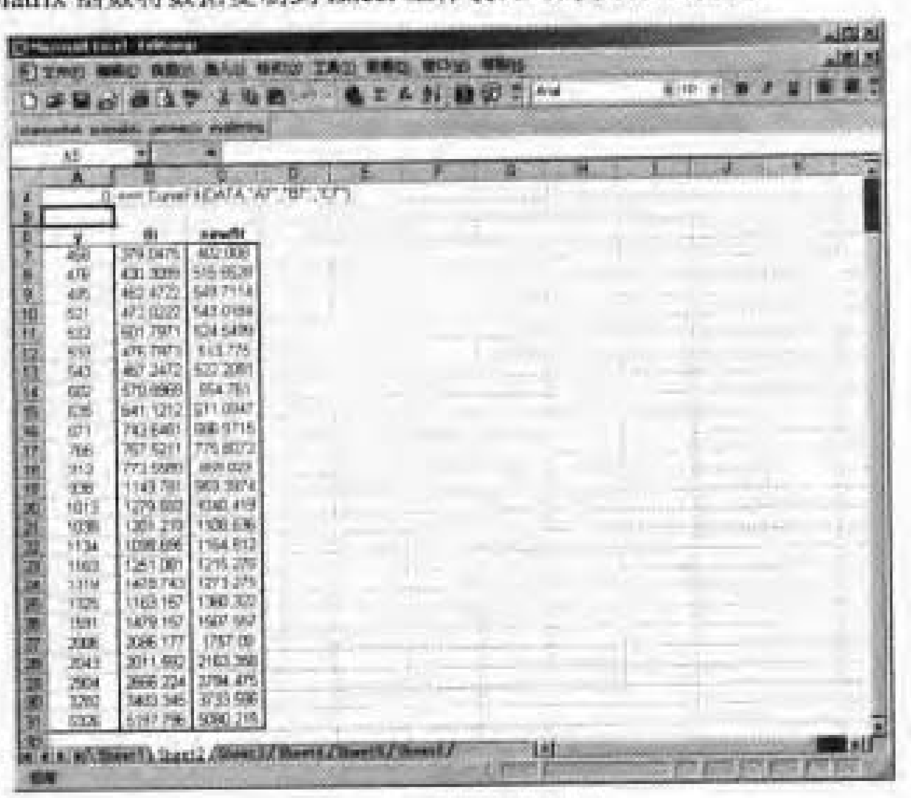


图 10-12　排序后的结果

示例完成以后，关闭图形窗口。

第 11 章　MATLAB 与 SPSS 接口

11.1　SPSS 软件

SPSS 软件是目前世界上有名的统计分析软件，它具有完全 Windows 风格的界面，在很少依赖命令操作的情况下可以实现数据导入、数据预处理、统计分析、绘图和报表输出等功能。该软件还有自己的 VBA 语言，可以在 COM 意义上与其他软件通信。软件的主要工作环境如图 11-1 所示。

图 11-1　SPSS 环境主窗口

11.2　SPSS 中的对象

在 SPSS 的较高版本中，可以通过内部提供的 SaxBasic 脚本语言结合自己原来的 Syntax 命令程序进行二次开发。SaxBasic 语言实际上是类似于 WordBasic、AccessBasic 以及 Excel、AutoCAD、MapInfo、GeoMedia 等应用程序中 Basic 语言的一种客户语言——VBA（Visual Basic for Application）。利用它，基于自动化机制，可以实现与同样支持该技术的其他专业应用程序之间的通信，如可以用 Word 来输出统计结果，可以与地理信息系统软件如 MapInfo, GeoMedia 之间进行数据传输和分析等。MATLAB 支持自动化机制，因此，通过 SPSS 提供的一系列对

象，MATLAB 可以与 SPSS 实现数据通信。SPSS 二次开发的主要编程环境——脚本编辑器如图 11-2 所示。

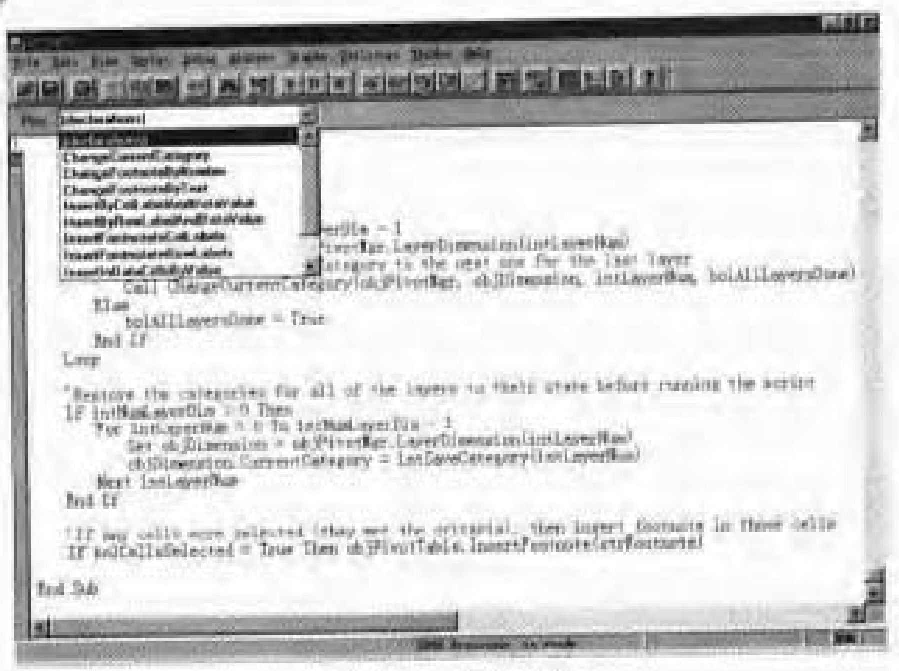


图 11-2　脚本编辑器

SPSS 中的对象及结构可以用图 11-3 表示。



图 11-3　SPSS 对象的树形结构

从图 11-3 中可以看出，并不是所有对象都在一个层次上，部分对象包含在其他对象中。位于树形图最顶端的是 Application 对象，它代表 SPSS 自身，所有其他对象都在 Application 对象之下。

第 2 层中包括 4 个对象：

Options 对象　　　利用该对象可以为输出浏览器、图表和数据等设置选项。

Documents 对象　　提供数据编辑器、语法文档窗口、输出浏览器和草稿文档窗口等四个窗口的属性和操作方法。

SPSS Info 对象　　提供 SPSS 的一些必要信息。

CS Application 对象　提供与网络有关的对象操作。

从图 11-3 中还可以看出，Documents 对象及其所包含的对象是所有对象中最重要的对象，它们几乎囊括了 SPSS 主要工作窗口中的所有工作内容。转轴表、图形和交互图是比较复杂的对象，各自又有很多子对象，从各个部位和方位来对父对象进行描述。

11.3 MATLAB 调用 SPSS

MATLAB 调用 SPSS 时是把 MATLAB 作为客户端应用程序来处理的。这时，利用 MATLAB 提供的 actxserver 函数，可以把 SPSS 程序作为自动化服务器打开。在 MATLAB 的命令窗口中键入：

```
spss=actxserver('spss.application')
```

其中，“spss.application”为 SPSS 的 ProgID 值。该语句的运行结果为：

```
spss =
    COM.spss.application
```

可见，SPSS 本身是一个 COM 对象。在命令行中键入

```
invoke(spss)
```

显示 SPSS 对象（指图 11-3 中的顶层对象——Application 对象）的所有方法名和方法参数，如下所示。其中，等号后面的小写单词表示方法返回值的类型，如 handle 表示句柄，string 表示字符串，void 表示没有返回值等。

```
OpenOutputDoc = handle OpenOutputDoc(handle, string)
OpenSyntaxDoc = handle OpenSyntaxDoc(handle, string)
OpenDataDoc = handle OpenDataDoc(handle, string)
NewOutputDoc = handle NewOutputDoc(handle)
NewSyntaxDoc = handle NewSyntaxDoc(handle)
NewDataDoc = handle NewDataDoc(handle)
GetDesignatedOutputDoc = handle GetDesignatedOutputDoc(handle)
GetDesignatedSyntaxDoc = handle GetDesignatedSyntaxDoc(handle)
ExecuteCommands = void ExecuteCommands(handle, string, bool)
ExecuteInclude = void ExecuteInclude(handle, string, bool)
IsBusy = bool IsBusy(handle)
Quit = void Quit(handle)
Interrupt = void Interrupt(handle)
SpssInfo = handle SpssInfo(handle)
OpenDocument = bool OpenDocument(handle, string, bool)
GetSPSSPath = string GetSPSSPath(handle)
ToolbarVisible = bool ToolbarVisible(handle, string)
```

OpenDraftDoc = handle OpenDraftDoc(handle, string)

NewDraftDoc = handle NewDraftDoc(handle)

GetDesignatedDraftDoc = handle GetDesignatedDraftDoc(handle)

ScriptParameter = string ScriptParameter(handle, int32)

我们试着调用其中的 OpenDataDoc 方法，打开一个 SPSS 数据文件 Employee data.sav，它位于 SPSS 软件的安装目录下。在命令窗口接着键入：

```
objDataDoc=OpenDataDoc(spss,'c:\Program Files\Spss\Employee data.sav')
```

结果如下所示：

```
objDataDoc =
    Interface.spss.application.OpenDataDoc
```

操作已经完成，但我们并没有看到打开的 SPSS 数据文件，用 get 函数查看对象的 Visible 属性，看该对象是不是隐藏起来了。在命令行键入

```
get(objDataDoc,'Visible')
```

结果为：

```
ans =
    0
```

可见，打开的数据文件确实是隐藏起来了，它的 Visible 属性值为 0。现在希望该数据文件可见，用 set 函数进行设置，即

```
set(objDataDoc,'Visible',1)
```

打开的数据文件马上就显示出来了，如图 11-4 所示。

图 11-4　显示打开的数据文件

介绍到这里，我们已经能够在 MATLAB 中获取和设置 SPSS 对象的属性，调用它们的方法了。所以，从理论上讲，你已经能够在不打开 SPSS 的情况下在 MATLAB 中享受 SPSS 的

服务了。这些服务项目包括导入数据、统计分析、绘制统计图、绘制美观的二维或三维交互图、报表输出，等等。

11.4 SPSS 调用 MATLAB

现在讨论 SPSS 调用 MATLAB 的问题，即 MATLAB 作为服务器端应用程序时的数据通信问题。与 MATLAB 调用 SPSS 一样，在方法上仍然沿用了前面 VB 与 MATLAB 通信的方法，因为它们都是在 ActiveX 自动化这个机制上进行通信的。所不同的是要面临 SPSS 这样一个“具体情况”。

SPSS 提供了一种称为 SaxBasic 的脚本语言，利用该语言实现对 MATLAB 的调用。

下面结合一个例子进行介绍。在 SPSS 中提供两个二元一次方程以后，调用 MATLAB 的“\”操作符求解方程组，然后在 SPSS 界面上显示方程组的解。

首先建立程序界面。SPSS 提供了一个比较简单的对话编辑器，如图 11-5 所示。利用该编辑器，可以像在 VB 中那样比较容易地设计图形用户界面。本例的界面如图 11-5 所示。在“方程 1”和“方程 2”方框中的文本框中赋值，分别指定两个方程的系数项和常数项。在下面的文本框中显示问题的解。对话框制作和 SaxBasic 语言语法等方面的细节内容这里不多谈，可参见 SPSS 文档或相关书籍。



图 11-5　SPSS 提供的用户对话编辑器界面

利用对话编辑器设计界面的工作完成以后，保存并退出，SPSS 会在脚本编辑器中自动生成相关代码，如图 11-6 所示。



图 11-6　自动生成对话框的代码

在脚本编辑器中继续完成下面的代码输入：

```
Dim objMATLAB As Object

Sub Main
Begin Dialog UserDialog 480,287,"求二元一次方程组的解",.DialogFunc
    PushButton 120,245,80,21,"确　定",.ok
    PushButton 280,245,80,21,"取　消",.cancel
    GroupBox 50,49,400,56,"方程 1",.GroupBox1
    TextBox 120,70,60,21,.txtF1a
    TextBox 240,70,60,21,.txtF1b
    TextBox 370,70,60,21,.txtF1c
    GroupBox 50,119,400,56,"方程 2",.GroupBox2
    TextBox 120,140,60,21,.txtF2a
    TextBox 240,140,60,21,.txtF2b
    TextBox 370,140,60,21,.txtF2c
    Text 50,14,300,14,"方程格式：a*x+b*y=c",.Text1
    TextBox 130,196,290,21,.txtAns
    Text 50,196,70,14,"方程的解",.Text2
    Text 80,77,20,14,"a",.Text3
    Text 200,77,20,14,"b",.Text4
    Text 330,77,30,14,"c",.Text5
    Text 80,147,20,14,"a",.Text6
    Text 200,147,20,14,"b",.Text7
    Text 330,147,20,14,"c",.Text8
End Dialog

Dim dlg As UserDialog
Dialog dlg

End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
  Select Case Action%
    Case 1 ' 对话框初始化
      DlgText "txtF1a","3"
      DlgText "txtF1b","5"
      DlgText "txtF1c","10"
      DlgText "txtF2a","8"
      DlgText "txtF2b","9"
      DlgText "txtF2c","52"
    Case 2 ' 控件的值改变或控件被单击
      Select Case DlgItem
        Case  "ok"
          Dim f1a As String
          Dim f1b As String
          Dim f1c As String
          Dim f2a As String
          Dim f2b As String
          Dim f2c As String
          Dim funsA As String
```

```
            Dim funsB As String
            Dim funs As String

            f1a=DlgText("txtF1a")
            f1b=DlgText("txtF1b")
            f1c=DlgText("txtF1c")
            f2a=DlgText("txtF2a")
            f2b=DlgText("txtF2b")
            f2c=DlgText("txtF2c")

            funsA="[" & f1a & " " & f1b & ";" & f2a & " " & f2b & "]"
            funsB="[" & f1c & "; " & f2c & "]"
            funs=funsA & "\" & funsB

            Set objMATLAB=CreateObject("matlab.application")
            DlgText "txtAns",CStr(objMATLAB.execute(funs))

          DialogFunc = True
        Case "cancel"
          Set objMATLAB=Nothing
          DialogFunc=False
      End Select
    End Select
  End Function
```

运行程序，在弹出的对话框中输入两个方程的参数，单击“确定”按钮，在“方程的解”文本框中显示方程组的解。如图 11-7 所示。

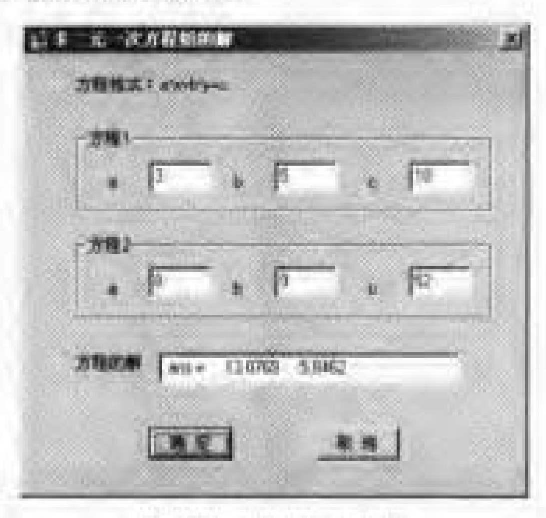


图 11-7　程序运行结果

运行上面代码以后，将打开 MATLAB 的命令窗口。注意，只有命令窗口，没有完整桌面。要显示完整桌面，在命令窗口键入下面的命令并回车即可。

```
desktop
```

如果在调用 MATLAB 的过程中不想打开它的界面，将 objMATLAB 对象的 Visible 属性值设置为 0 就可以了，即

```
objMATLAB.Visible=0
```

第 12 章　COM 生成器（COM Builder）

从 6.5 版本开始，MATLAB 提供了 COM 生成器。COM 生成器提供了实现 MATLAB 独立应用的一种新途径。它能把 MATLAB 开发的算法做成组件，这些组件作为独立的 COM 对象，可以直接被 Visual Basic, Visual C++或其他支持 COM 的语言所引用。本章结合 Visual Basic 语言介绍 MATLAB 的 COM 生成器的使用方法。通过创建简单的代码模块，可以将 COM 生成器组件简单地集成到 Visual Basic 工程中去，代码模块中的函数和/或过程装载必要的组件，调用必要的方法并处理所有错误。

12.1　创建 COM 生成器组件

用 MATLAB COM 生成器创建 COM 组件是一个简单的过程，只需要 4 个步骤，即创建工程，管理 M 文件和 MEX 文件、生成组件、打包和分发组件。

12.1.1　创建工程

在 MATLAB 命令行中输入命令 comtool，打开“MATLAB COM Builder”对话框，如图 12-1 所示。它是 MATLAB COM 生成器的主要工作环境。

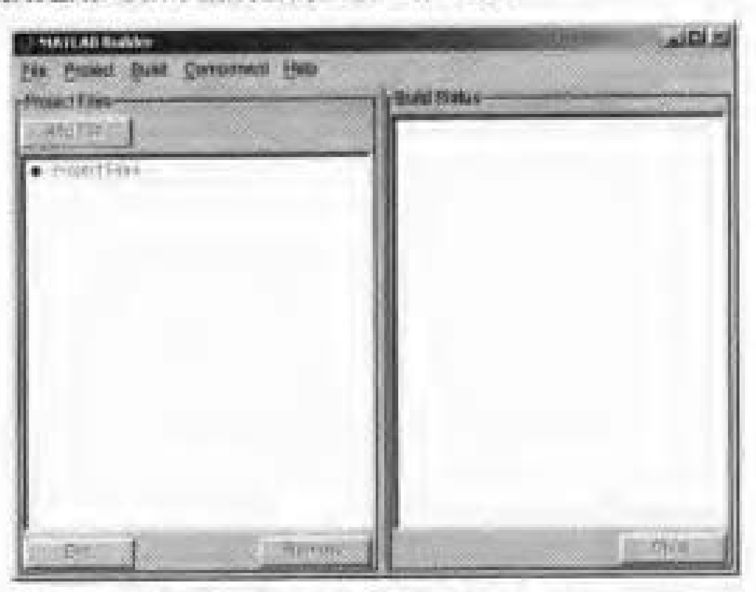


图 12-1　MATLAB COM 生成器主窗口

在“File”菜单中选择“New Project”选项，打开“New Project Settings”对话框，如图 12-2 所示。

在“Component name”文本框中输入组件（DLL 文件）的名称。输入组件名以后，生成器自动在“Class name”文本框中输入与组件名相同的名称。用户可以根据需要进行修改。

注意：*尽管组件名和类名可以相同，但组件名不能与任何 M 文件或 MEX 文件的文件名相同。*

若要把类添加到组件中，则在“Class name”文本框中输入类名，并单击“Add>>”按钮。添加的类将显示在“Classes”列表框中。

在“Project version”文本框中设置组件的版本号，默认时组件版本号为 1.0。

“Project directory”文本框指定在编译和打包模型时，工程和相关文件的存放位置。工程目录由当前目录和组件名自动组合生成。

如果选择“Create a singleton MCR”核选框，使用组件时只创建一个 MCR 实例。

可以创建编译模型的一个调试版本，并能在调用 MATLAB 编译器时指定详细输出，该调试版本一方面允许跟踪，使出错报告显示 M 文件和发生错误的行。所有跟踪信息都会进行报告。如果不进行调试，则得不到 MATLAB 代码中发生错误的位置的指示。另一方面，它允许使用 Visual Studio 调试器进行完整的调试。

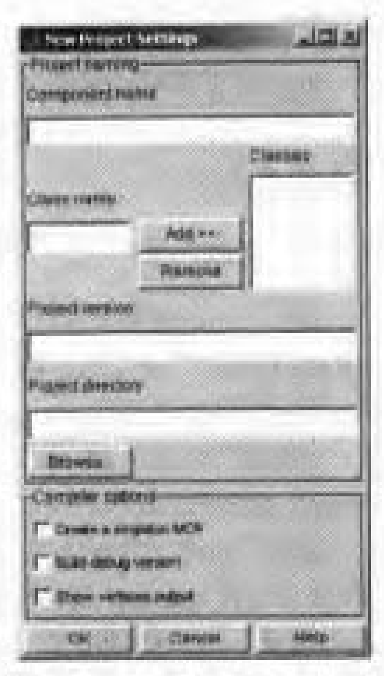


图 12-2 “New Project Settings”对话框

设置完以后，单击“OK”按钮，它们就成为工程工作空间的一部分并与添加到工程中的所有 M 文件和 MEX 文件一起被保存到工作空间中。一个名为<component_name>.cbl 的工程文件被自动添加到工程目录中。

12.1.2 管理 M 文件和 MEX 文件

创建工程以后，生成器主窗口中的“Project”、“Build”和“Component”等 3 个菜单就变为可用，如图 12-3 所示。

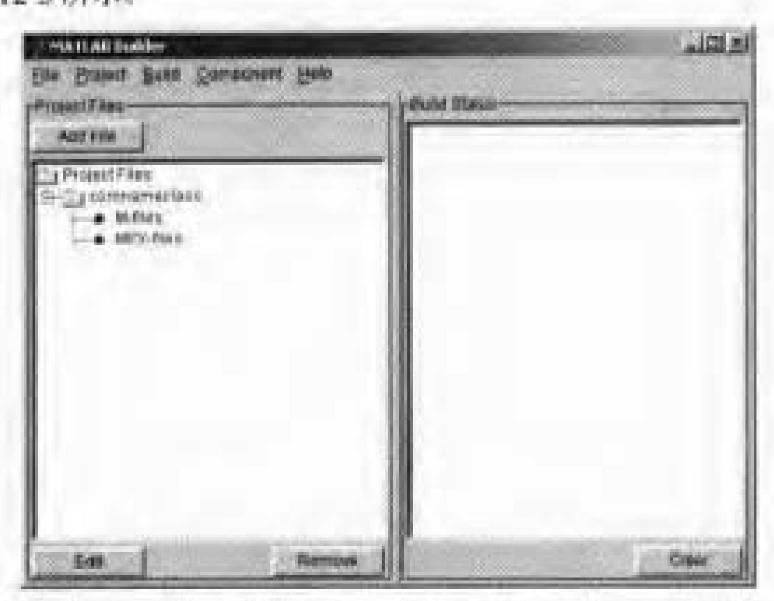


图 12-3 选项被激活的主窗口

单击“Add File”按钮或从“Project”菜单中选择“Add File...”选项，可向工程中添加 M 文件和/或 MEX 文件。一次只能添加一个文件。

单击“Remove”按钮或从“Project”菜单中选择“Remove”选项，可删除所有选定的 M

文件或 MEX 文件。可以选择多个文件，然后将其一次删除。

选择 M 文件以后单击“Edit”按钮，或从“Project”菜单中选择“Edit File...”选项，或直接双击 M 文件名，都可以在 MATLAB 编辑器中打开该 M 文件并进行编辑和调试。但不能编辑 MEX 文件。

12.1.3 生成组件

定义工程设置并添加必要的 M 函数和 MEX 函数以后，可以通过选择“Build”菜单中的“COM Object”选项来调用 MATLAB 编译器，把中间源文件写到<project_dir>\scr 目录中，将必要的输出文件写到<project_dir>\distrib 目录中。

“Build Status”面板显示生成过程中的输出，通知遇到的任何问题。出现在<project_dir>\distrib 目录中的将是一个 DLL 文件。生成的 DLL 文件自动注册到系统中。

选择“Build”菜单中的“Clear Status”选项，清除“Build Status”面板。生成过程的输出保存在文件<project_dir>\build.log 中。要打开该日志文件，在“Build”菜单中选择“Open Build Log”选项。该文件提供了一个生成过程的记录，在清除“Build Status”面板上的内容以后，可以参见该记录。

12.2 利用 COM 生成器组件编程

12.2.1 给 COM 生成器组件对象添加方法和属性

COM 生成器自动将 M 文件中的全局变量转换为属性。全局变量是用 global 关键字声明的变量。当有一个经常使用，但很少变化的大数组时，将它设置为属性很有用。

下面的例子演示在矩阵因式分解类中使用类属性。本例开发一个类，对同一个矩阵进行乔累斯基分解、LU 分解和 QR 分解。示例中把输入矩阵作为类属性保存，这样，它不必传递到因式分解过程。

考虑下面 3 个 M 文件：

```
Cholesky.m
function [L] = Cholesky()
    global A;
    if (isempty(A))
        L = [];
        return;
    end
    L = chol(A);

LUDecomp.m
function [L,U] = LUDecomp()
    global A;
    if (isempty(A))
        L = [];
        U = [];
        return;
    end
    [L,U] = lu(A);
```

```
QRDecomp.m
function [Q,R] = QRDecomp()
    global A;
    if (isempty(A))
        Q = [];
        R = [];
        return;
    end
    [Q,R] = qr(A);
```

这 3 个文件有一个公共的全局变量 A，每个函数对 A 进行矩阵因式分解，并返回结果。

要生成类，创建一个新的名为 mymatrix 的 COM 生成器工程，版本为 1.0。添加一个名为 myfactor 的类到组件中，把上面 3 个 M 文件添加到类中并生成组件。

使用下面的 Visual Basic 过程测试 myfactor 类。首先，在 Visual Basic 主菜单中单击“工程”菜单中的“引用...”选项，利用“浏览...”按钮找到并选择刚生成的组件。然后运行下面的过程。它创建一个 myfactor 类的实例，并给属性 A 指定一个 Double 型矩阵。最后，调用 3 个因式分解方法。

```
Sub TestFactor()
    Dim x(1 To 2, 1 To 2) As Double
    Dim C As Variant, L As Variant, U As Variant, _
    Q As Variant, R As Variant
    Dim factor As myfactor

    On Error GoTo Handle_Error
    Set factor = New myfactor
    x(1, 1) = 2#
    x(1, 2) = -1#
    x(2, 1) = -1#
    x(2, 2) = 2#
    factor.A = x
    Call factor.cholesky(1, C)
    Call factor.ludecomp(2, L, U)
    Call factor.qrdecomp(2, Q, R)
    Exit Sub
Handle_Error:
    MsgBox (Err.Description)
End Sub
```

12.2.2 给 COM 生成器组件对象添加事件

通过简单的语言标注，COM 生成器支持事件或回调。可以简单地提供一个 MATLAB 函数，将其作为事件的原型。然后，在客户代码（Visual Basic、C++等）中提供一个该函数的实现。

通过将%#event 标注放置到代码中，可以把 MATLAB 函数转换为事件函数。当把相同的函数作为 COM 生成器对象的方法时，编译器为该方法生成一个外调接口，并把该方法当成事件。这个外调接口由客户代码实现。

在代码中使用回调的一些例子包括：

- 在长时间的计算中，给客户应用周期性的回调。例如：如果一个需要 n 次迭代的任务，可以通过事件处理，在进度条中为每次迭代做出显示；
- 在计算还没完成却进行后续操作时给出警告；
- 给出计算过程中的当前结果，并继续计算。

下面的例子在连接进度条的过程中使用回调。MATLAB 函数 iterate 进行 n 次迭代，并每隔 inc 次迭代激发一次事件。当函数结束时，它返回一个单一的输出。为了模拟实际行为，在主循环中设置一次暂停。这样，每次迭代时函数等待 1 秒钟。

MATLAB 函数 iterate.m 和 progress.m 的代码如下所示：

iterate.m

```
function [x] = iterate(n,inc)
    %初始化 x
    x = 0;
    % 进行 n 次迭代，每隔 inc 次进行一次回调。
    k = 0;
    for i=1:n
        k = k + 1;
        if k == inc
            progress(i);
            k = 0;
        end;
        % 对 x 作一些操作
        x = x + 1;
        % 暂停 1 秒，假设在做某件事情
        pause(1);
    end;
```

progess.m

```
function progress(i)
    %#event
    i
```

iterate 函数进行 n 次迭代，并每隔 inc 次迭代调用一次 progress 函数，将当前迭代次数作为一个变量进行传递。本函数在 MATLAB 中运行时，progress 函数每次被调用时显示 i 的值。假设创建了一个 COM 生成器组件，它有两个作为类方法的函数，本例假设组件具有一个名为 myclass 的类，生成的 COM 类有一个 iterate 方法和一个 progress 事件。本例中事件句柄的 VB 语法为：

```
Sub aClass_progress(Byval I As Variant)
```

其中，aClass 是一个用于实例的变量名。事件函数的所有输入参数都使用 ByVal 关键字。为了使过程可用，用 withEvents 关键字定义 aClass 变量。

本例基于一个简单的 VB 窗体进行讨论，窗体中有 3 个 TextBox 控件，一个 CommandButton 控件和一个 ProgressBar 控件。第 1 个文本框 Text1 输入迭代次数，保存在窗体级变量 N 中，第 2 个文本框 Text 2 输入回调增量，保存在变量 Inc 中，第 3 个文本框 Text 3 显示函数输出。单击命令按钮 Command1，运行类的 iterate 方法。进度条控件 Progress Bar1 在 progress 事件发生时进行更新。

```
'窗体级变量
Private WithEvents aClass As myclass                '类实例
Private N As Long                                   '迭代次数
Private Inc As Long                                 '回调增量
Private Sub Form_Load()
'装载窗体时，创建新的 myclass 实例
    Set aClass = New myclass
    '初始化变量
    N = 2
    Inc = 1
End Sub
Private Sub Text1_Change()
'当 Text1 文本框中发生改变时更新 N 的值
    On Error Resume Next
    N = CLng(Text1.Text)
    If Err <> 0 Then N = 2
    If N < 2 Then N = 2
End Sub
Private Sub Text2_Change()
'当 Text2 文本框中发生改变时更新 Inc 的值
    On Error Resume Next
    Inc = CLng(Text2.Text)
    If Err <> 0 Then Inc = 1
    If Inc <= 0 Then Inc = 1
End Sub
Private Sub Command1_Click()
'单击“Execute”按钮时运行函数
    Dim x As Variant
    On Error GoTo Handle_Error
    '初始化进度条
    ProgressBar1.Min = 1
    ProgressBar1.Max = N
    Text3.Text = ""
    '迭代 N 次，每间隔 Inc 次进行回调
    Call aClass.iterate(1, x, CDbl(N), CDbl(Inc))
    Text3.Text = Format(x)
    Exit Sub
Handle_Error:
    MsgBox (Err.Description)
End Sub
Private Sub aClass_progress(ByVal i As Variant)
'事件句柄，iterate 函数每次调用 progress 函数时被调用。
'进度条随传入的值更新，并前进
    ProgressBar1.Value = i
End Sub
```

12.2.3 创建类实例

调用类的方法以前，必须创建一个包含方法的类的实例，VBA 提供了两个技巧来做这件事情。

- CreateObject 函数；
- New 运算符。

1．CreateObject 函数

本方法使用 Visual Basic 应用程序接口（API）的 CreateObject 函数创建类的实例，要使用本方法，需要定义一个 Object 类型的变量作为类实例的引用，并将组件的 ProgID 作为变量调用 CreateObject 函数，如下例所示：

```
Function foo(x1 As Variant, x2 As Variant) As Variant
    Dim aClass As Object

    On Error Goto Handle_Error
    aClass = CreateObject("mycomponent.myclass.1_0")
    '调用 aClass 的某些方法
    Exit Function
Handle_Error:
    foo = Err.Description
End Function
```

2．New 运算符

本方法使用 New 运算符创建类的实例，在使用本法以前，必须使用当前 VBA 工程中包含该类的类型库。要做到这一点，在 Visual Basic 编辑器中选择“Tools”菜单，然后选择“References...”选项，显示“Available References”对话框。从该对话框中选择必要的类型库。

下面的例子演示如何使用 New 运算符创建一个类实例，假设在调用本函数以前从“Available References”列表框中选择了“mycomponent 1.0 Type Library”选项。

```
Function foo(x1 As Variant, x2 As Variant) As Variant
    Dim aClass As mycomponent.myclass
    On Error Goto Handle_Error
    Set aClass = New mycomponent.myclass
    '(调用 aClass 的一些方法)
    Exit Function
Handle_Error:
    foo = Err.Description
End Function
```

本例中，类实例定义为 myclass，完整的表达形式为<component-name>.<class-name>，它保证在当前工程的其他库中包含名为 myclass 的类时不会出现名称冲突。

两种方法相比较，第 1 种方法不需要引用 VBA 工程的类库，使用更灵活，但速度慢；第 2 种方法可以获得更快的运行效果。

在前面的两个例子中，用于方法调用的类型是一个过程的局部变量，它为每次调用创建和拆卸一个新的类实例，一个替代方法是声明一个模块级类实例，它可以被所有的函数调用和重复使用，就像前面例子的初始代码中一样。

```
Dim aClass As mycomponent.myclass

Function foo(x1 As Variant, x2 As Variant) As Variant
    On Error Goto Handle_Error
    If aClass Is Nothing Then
```

```
            Set aClass = New mycomponent.myclass
        End If
        ' (调用 aCalss 的一些方法)
        Exit Function
    Handle_Error:
        foo = Err.Description
    End Function
```

12.2.4 调用类实例的方法

创建类实例以后，可以调用类的方法来获取编译过的 MATLAB 函数。

当方法有输出变量时，第 1 个变量总是 nargout，它的数据类型为 Long。这个输入参数传递 MATLAB nargout 参数给编译函数，并且指定要求有多少个输出变量。没有输出变量的方法不传递 nargout 变量。nargout 变量是输出参数，列在原始 MATLAB 函数的左侧，紧接着的输入参数列在右侧，所有输入和输出变量都是 Variant 类型的。该类型为默认的 VB 数据类型。

一般地，除了用户自定义类型外，可以给类的方法提供任何 VB 类型作为变量。

12.2.5 处理 varargin 和 varargout 变量

当原始 MATLAB 函数中存在 varargin 和/或 varargout 时，这些参数将作为列表中最后的输入/输出参数被添加到类方法的变量列表中，可以通过创建一个 Variant 型数组，把数组的每个元素指定给各自的输入变量并作为 Variant 型数组传递给多个变量。

下面的例子创建一个 varargin 数组，以调用一个形如 y=foo(varargin)的 MATLAB 函数生成的方法。

```
Function foo(x1 As Variant, x2 As Variant, x3 As Varaint, _
                    x4 As Variant, x5 As Variant) As Variant
    Dim aClass As Object
    Dim v(1 To 5) As Variant
    Dim y As Variant

    On Error Goto Handle_Error
    v(1) = x1
    v(2) = x2
    v(3) = x3
    v(4) = x4
    v(5) = x5
    aClass = CreateObject("mycomponent.myclass.1_0")
    Call aClass.foo(1,y,v)
    foo = y
    Exit Function
Handle_Error:
    foo = Err.Description
End Function
```

MWComUtil 工具库中的 MWUtil 类提供 MWPack 帮助函数，以创建 varargin 参数。

12.2.6 在调用方法的过程中控制错误

创建类实例或调用类方法时如果发生错误，则在当前过程中创建一个出错标记。VB 在声明 On Error Goto <label>的过程中提供一个错误控制兼容性，发生错误时跳至<label>行，所有的错误都通过这种方式来控制，包括原始 MATLAB 代码中的错误。

12.2.7 修改标记

每个 MATLAB Excel 生成器组件提供一个 MWFlags 类型的名为 MWFlags 的单一读写属性。MWFlags 属性由两部分组成：数组格式化标记和数据转换标记。数据转换标记改变数据从 Variant 型向 MATLAB 类型转换或其逆转换时的行为。默认时，这些类型提供了它自己的 MWFlags 属性，忽略了方法被调用的类的属性。特别是在特定变量的设置与类属性的默认设置不同的情况下，提供 MWArg 类。

本小节讨论如何设置这些标记。

1．数组格式化标记

数组格式化标记引导数据转换，以生成一个源于一般 Variant 型数据的 MATLAB 单元数组或矩阵，并作为输入参数，或生成一个 Variant 型数组或包含基本类型数组的单一 Variant 数据作为输出参数。

2．数据转换标记

数据转换标记处理数组元素的类型转换。CoerceNumericToType 和 InputDateFormat 两个数据转换标记，控制数值和日期类型如何从 VBA（Visual Basic for Application）转换到 MATLAB。

考虑下面的例子：

```
Sub foo( )
    Dim aClass As mycomponent.myclass
    Dim var1, var2 As Variant
    Dim y As Variant

    On Error Goto Handle_Error
    var1 = 1
    var2 = 2#
    Set aClass = New mycomponent.myclass
    Call aClass.foo(1,y,var1,var2)
    Exit Sub
Handle_Error:
    MsgBox(Err.Description)
End Sub
```

本例将 Variant/Integer 型的 var1 转换为 int16 型，将 Variant/Double 型的 var2 转换为 Double 型。如果原始的 MATLAB 函数希望有两个 Double 型变量作为输入，则上面的代码导致错误。解决的方法是将 var1 指定为 Double 型。但这可能很难办到。此时，将 CoerceNumericToType 标记设为 mwTypeDouble，让数据转换器将所有的数值输入转换为 Double 型，在前面的例子中，创建类以后，调用方法前放置下面的代码行。

```
aClass .MWFlags.DataConversionFlags.CoerceNumericToType = mwTypeDouble
```

InputDateFormat 标记控制 VBA Date 型数据的转换方式。本例将当前的日期和时间作为一个输入变量进行传递，并转换为字符串。

```
Sub foo( )
    Dim aClass As mycomponent.myclass
    Dim today As Date
    Dim y As Variant

    On Error Goto Handle_Error
    today = Now
    Set aClass = New mycomponent.myclass
    aClass. MWFlags.DataConversionFlags.InputDateFormat =
    mwDateFormatString
    Call aClass.foo(1,y,today)
    Exit Sub
Handle_Error:
    MsgBox(Err.Description)
End Sub
```

12.3 应用举例

本例首先创建 3 个 M 文件，分别实现用 Jacobi 迭代法、Gauss-Seidel 迭代法和 SOR（超松弛）迭代法求解线性方程组。然后用它们创建 COM 组件，并在 VB 中利用该组件进行线性方程组求解。

12.3.1 创建 M 文件

首先创建 M 文件，如下所示。其中参数 a,b,x0 和 eps 均为公共变量。

Jacobi.m:

```
function s=jacobi()
  % Jacobi 迭代法解线性方程组
  % a 为系数矩阵，b 为方程组 ax=b 的右端项，x0 为初值
  global a b x0 eps;
  b=b';
  x0=x0';
  D=diag(diag(a)); %求对角矩阵
  D=inv(D); %求对角矩阵的逆
  L=tril(a,-1); %求严格下三角矩阵
  U=triu(a,1); %求严格上三角矩阵
  B=-D*(L+U);
  f=D*b;
  s=B*x0+f;
  while norm(s-x0)>=eps
      x0=s;
      s=B*x0+f;
  end
  s=s';
  return
```

Gauss.m：

```
function s=Gauss()
  % Gauss-Seidel 迭代法解线性方程组
  % a 为系数矩阵，b 为方程组 ax=b 的右端项，x0 为初值
  global a b x0 eps;
  b=b';
  x0=x0';
  D=diag(diag(a)); %求对角矩阵
  L=tril(a,-1); %求严格下三角矩阵
  U=triu(a,1); %求严格上三角矩阵
  C=inv(D+L);
  B=-C*U;
  f=C*b;
  s=B*x0+f;
  while norm(s-x0)>=eps
      x0=s;
      s=B*x0+f;
  end
  s=s';
  return
```

SOR.m：

```
function s=SOR(w)
  % SOR(超松弛)迭代法解线性方程组
  % a 为系数矩阵，b 为方程组 ax=b 的右端项
  % x0 为初值，w 为松弛因子
  global a b x0 eps;
  b=b';
  x0=x0';
  D=diag(diag(a)); %求对角矩阵
  L=-tril(a,-1); %求严格下三角矩阵
  U=-triu(a,1); %求严格上三角矩阵
  C=inv(D-w*L);
  B=C*[(1-w)*D+w*U];
  f=w*C*b;
  s=B*x0+f;
  while norm(s-x0)>=eps
      x0=s;
      s=B*x0+f;
  end
  s=s';
  return
```

12.3.2 创建 COM 生成器组件

在 MATLAB 命令窗口中输入命令 comtool，启动 MATLAB COM 生成器图形用户界面。在“File”菜单中选择“New Project”选项，打开“New Project Settings”对话框，如图 12-4 所示。

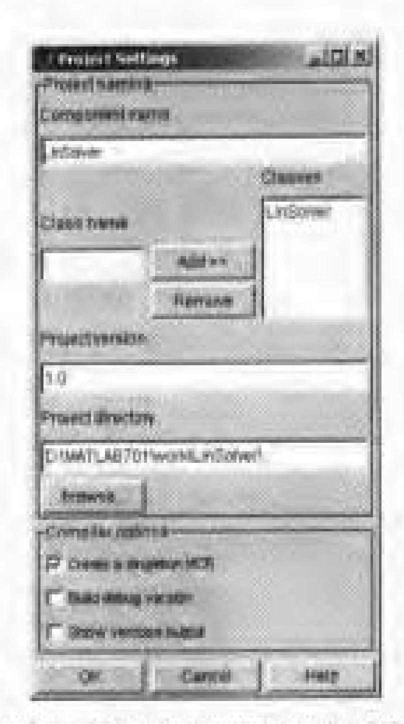


图 12-4 "New Project Settings" 对话框

在 "New Project Settings" 对话框中，按下面各步骤进行设置：

（1）"在 Component name" 文本框中输入组件名 "LinSolver"，单击 Tab 键将光标移动到 "Class name" 文本框中。

（2）删除自动命名的类名 LinSolverClass，在 "Class name" 文本框中输入类名 "LinSolver"，单击 "Add" 按钮。

（3）默认时的版本号为 1.0，不改变版本号。

（4）"Project directory" 文本框中包含一个默认的路径组合，其中包括 COM 生成器的启动路径和组件名。可以将这个路径改变为任何其他路径。如果选择的路径不存在，系统会提示你是不是要创建该路径。

（5）单击 "OK" 按钮，回到 COM 生成器主窗口。

然后按照以下步骤生成 COM 组件：

（1）在 COM 生成器图形用户界面中单击 "Add File" 按钮。

（2）在前面创建的 3 个 M 文件的保存路径中找到对应文件，单击它，并单击 "Open" 按钮。

（3）重复（1）、（2），将 3 个 M 文件全部添加到模型中去。

（4）在 "Build" 菜单中选择 "COM Object" 选项，生成 COM 组件。

成功创建 COM 组件以后，创建信息显示在 "Build Status" 面板中，如图 12-5 所示。

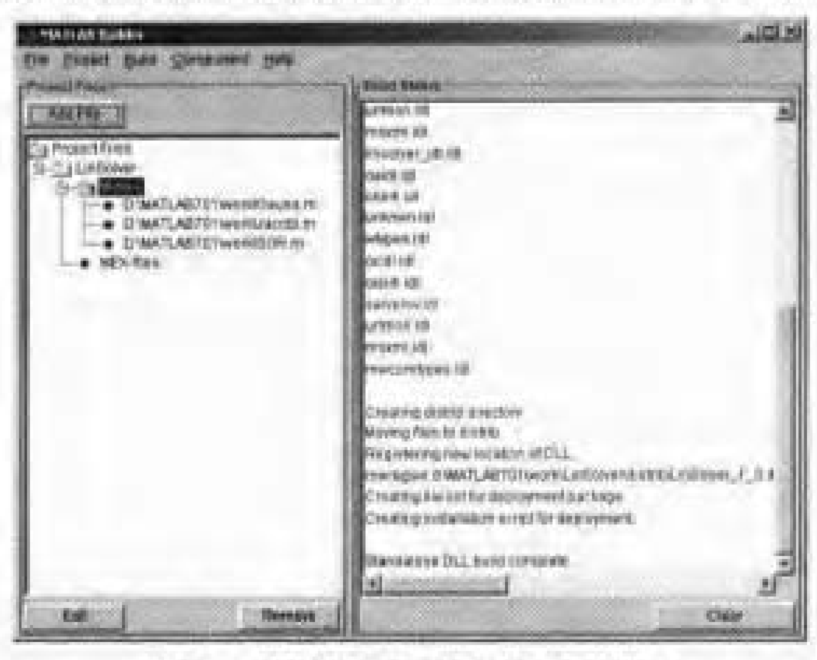


图 12-5　成功创建 COM 组件以后显示的信息

12.3.3　在 Visual Basic 中使用 COM 组件

用 COM 生成器完成 COM 组件的创建以后，创建 Visual Basic 程序，在程序中使用该 COM 组件。

1．创建 Visual Basic 工程

下面以 Visual Basic 6.0 为例，创建一个 Visual Basic 工程，其步骤如下：

（1）启动 Visual Basic。

（2）在“新建工程”对话框中选择“标准 EXE”选项，确定工程的类型。单击“打开”按钮，创建一个新的 Visual Basic 工程。

（3）在主菜单中，选择工程→引用，显示“引用”对话框。

（4）“引用”对话框中列出了所有可用的组件，在其中选择“LinSolver 1.0 Type Library”选项。单击“确定”按钮，返回到 Visual Basic 主菜单。

2．创建用户界面

现在添加一系列控件到空白的窗体上，完成对话框的创建。添加控件以后的窗体如图 12-6 所示。各控件的属性设置如表 12-1 所示。

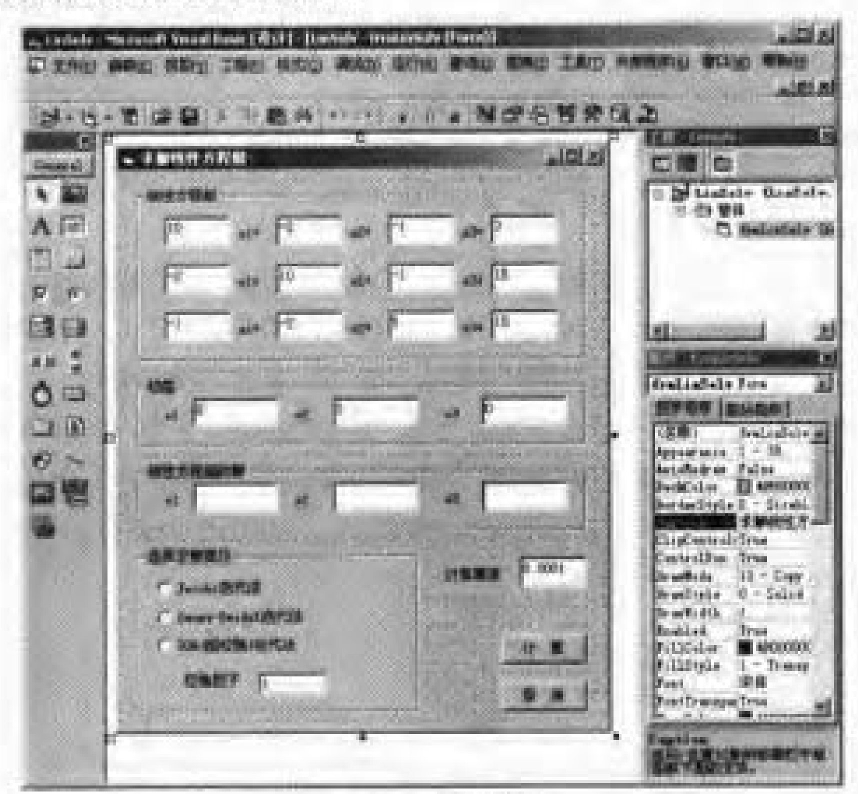


图 12-6 窗体设计

表 12-1 控件属性设置

对象类型	名称	其他属性设置
Form	frmLinSolv	Caption=求解线性方程组
Frame	fmeFun	Caption=线性方程组
	fmeInit	Caption=初值
	fmeResult	Caption=线性方程组的解
	fmeSolver	Caption=选择求解算法
TextBox	txtFun	Index=0 Text=10
		Index=1 Text=-2
		Index=2 Text=-1
		Index=3 Text=3

续表

对象类型	名称	其他属性设置
TextBox	txtFun	Index=4　Text=-2
		Index=5　Text=10
		Index=6　Text=-1
		Index=7　Text=15
		Index=8　Text=-1
		Index=9　Text=-2
		Index=10　Text=5
		Index=11　Text=10
	txtInit	Index=0　Text=0
		Index=1　Text=0
		Index=2　Text=0
	txtResult	Index=0　Text=""
		Index=1　Text=""
		Index=2　Text=""
	txtWeight	Text=1　Enabled=False
	txtPrec	Text=0.0001
Label	lblFun	Index=0　Caption=x1+
		Index=1　Caption=x2+
		Index=2　Caption=x3=
		Index=3　Caption=x1+
		Index=4　Caption=x2+
		Index=5　Caption=x3=
		Index=6　Caption=x1+
		Index=7　Caption=x2+
		Index=8　Caption=x3=
	lblInit	Index=0　Caption=x1
		Index=1　Caption=x2
		Index=2　Caption=x3
	lblResult	Index=0　Caption=x1
		Index=1　Caption=x2
		Index=2　Caption=x3
	txtWeight	Caption=松弛因子　Enabled=False
	txtPrec	Caption=计算精度
OptionButton	optJacobi	Caption=Jacobi 迭代法
	optGauss	Caption=Gauss-Seidel 迭代法
	optSOR	Caption=SOR(超松弛)迭代法
CommandButton	cmdCompute	Caption=计算
	cmdCancel	Caption=取消

3. 添加代码

窗体和控件属性设置完毕以后，添加对象代码。这些代码将引用表 12-1 中的控件名和属性值。如果要给任何控件或属性指定不同的名称或值，需要通过改变代码来体现这些改变。下面的代码利用前面创建的 LinSolver 组件求解给定的线性方程组，并显示计算结果。

```
Option Explicit

Private Enum Method
    jacobi = 1
    gauss = 2
    sor = 3
End Enum

Private mlsvSolver As LinSolver.LinSolver
Private mdblIndex(1 To 3, 1 To 3) As Double
Private mdblRight(1 To 3) As Double
Private mdblInit(1 To 3) As Double
Private mdblPrec As Double
Private mdblWeight As Double
Private mmtdMethod As Method

Private Sub cmdCancel_Click()
    '卸载窗体
    Unload frmLinSolv
End Sub

Private Sub Form_Load()
    '创建 LinSolver 类的实例
    Set mlsvSolver = New LinSolver.LinSolver
    '默认时选择 Jacobi 迭代法进行计算
    optJacobi.Value = True
    mmtdMethod = jacobi
End Sub

Private Sub cmdCompute_Click()
    Dim varResult As Variant

    '系数矩阵
    mdblIndex(1, 1) = Val(txtFun(0).Text)
    mdblIndex(1, 2) = Val(txtFun(1).Text)
    mdblIndex(1, 3) = Val(txtFun(2).Text)
    mdblIndex(2, 1) = Val(txtFun(4).Text)
    mdblIndex(2, 2) = Val(txtFun(5).Text)
    mdblIndex(2, 3) = Val(txtFun(6).Text)
    mdblIndex(3, 1) = Val(txtFun(8).Text)
    mdblIndex(3, 2) = Val(txtFun(9).Text)
    mdblIndex(3, 3) = Val(txtFun(10).Text)

    '右端项
    mdblRight(1) = Val(txtFun(3).Text)
    mdblRight(2) = Val(txtFun(7).Text)
    mdblRight(3) = Val(txtFun(11).Text)

    '初值
    mdblInit(1) = Val(txtInit(0).Text)
```

```
    mdblInit(2) = Val(txtInit(1).Text)
    mdblInit(3) = Val(txtInit(2).Text)

    '精度
    mdblPrec = Val(txtPrec.Text)

    '选择不同的方法进行求解
    Select Case mmtdMethod
        Case jacobi
            mlsvSolver.a = mdblIndex
            mlsvSolver.b = mdblRight
            mlsvSolver.x0 = mdblInit
            mlsvSolver.eps = mdblPrec
            Call mlsvSolver.jacobi(1, varResult)
        Case gauss
            mlsvSolver.a = mdblIndex
            mlsvSolver.b = mdblRight
            mlsvSolver.x0 = mdblInit
            mlsvSolver.eps = mdblPrec
            Call mlsvSolver.gauss(1, varResult)
        Case sor
            mlsvSolver.a = mdblIndex
            mlsvSolver.b = mdblRight
            mlsvSolver.x0 = mdblInit
            mlsvSolver.eps = mdblPrec
            mdblWeight = Val(txtWeight.Text)
            Call mlsvSolver.sor(1, varResult, mdblWeight)
    End Select

    '输出计算结果
    Dim intI As Integer
    For intI = 1 To 3
        txtResult(intI - 1).Text = ""
        txtResult(intI - 1).Text = Str(varResult(1, intI))
    Next
End Sub

Private Sub optGauss_Click()
    mmtdMethod = gauss
    lblWeight.Enabled = False
    txtWeight.Enabled = False
End Sub

Private Sub optJacobi_Click()
    mmtdMethod = jacobi
    lblWeight.Enabled = False
    txtWeight.Enabled = False
End Sub

Private Sub optSOR_Click()
```

```
        mnudMethod = sor
        lblWeight.Enabled = True
        txtWeight.Enabled = True
    End Sub
```

选择"文件"菜单中的"保存工程"选项，保存工程。主窗体名称为 frmLinSolv，并输入"LinSolv.vbp"作为工程名。

默认时程序求解下面的方程组，3 个变量的初值都是 0。

$$
\begin{aligned}
10x_1 - 2x_2 - x_3 &= 3 \\
-2x_1 + 10x_2 - x_3 &= 15 \\
-x_1 - 2x_2 + 5x_3 &= 10
\end{aligned}
$$

4．运行程序

按下 F5 键，运行程序。运行结果如图 12-7 所示。

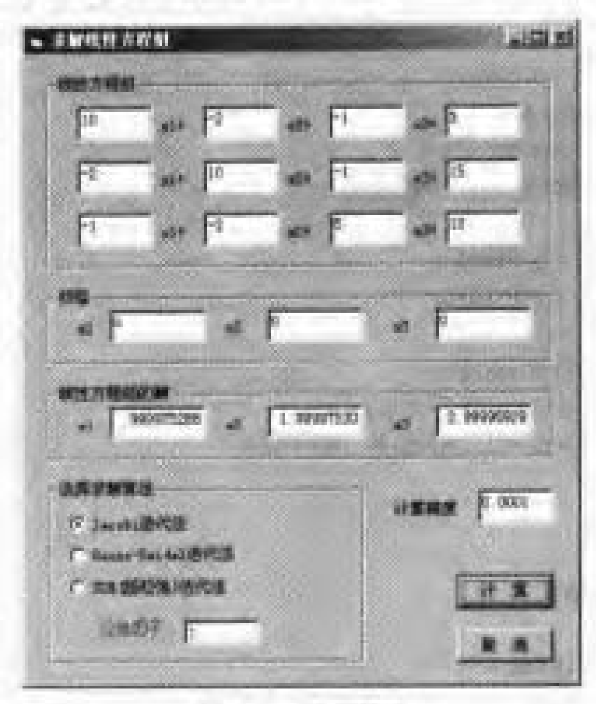


图 12-7　运行结果

12.4　COM 组件的部署

部署 COM 组件指的是将 COM 组件安装到没有安装 MATLAB 的其他机器上。进行 COM 组件部署，需要将组件打包分发到目标机器，并在目标机器上安装 MRC。

12.4.1　组件打包

模型编译和 COM 对象的测试都成功以后，就可以将组件进行打包，以便分发给终端用户。打包时会将组件和它的所有支持库打包到一个自解压可执行程序中。要打包组件，按照以下步骤进行。

（1）返回到 COM 生成器主界面，如果已经关闭，重新启动它并载入模型工程。

（2）按顺序选择 Component→Package Component 选项，打开"Package Files"对话框，

如图 12-8 所示。



图 12-8 “Package Files” 对话框

选择对话框中的“Include MCR”时，包括 MCRInstaller.exe 文件。单击“Create...”按钮，创建一个名为<组件名>.exe 的自解压可执行程序，其中包含表 12-2 中列出的文件。

表 12-2 组件打包时包含的文件

文 件 名	描 述
_install.bat	自解压程序运行的脚本程序
<componentname_projectversion>.dll	编译后的组件
MCRInstaller.exe	自解压 MATLAB 组件运行时库工具，与平台有关系
<componentname>.ctf	组件技术文档，与平台有关

要将组件安装到另一台计算机上，需要将<组件名>.exe 包复制到该计算机上，并从命令行中运行它。

12.4.2 MCR

使用表 12-2 中的 MCRInstaller.exe 程序在目标机器上安装 MCR。MCR 安装组件正常运行所必需的库文件。第 3 章比较详细地介绍了 MCR 的基本情况及其安装方法。

12.4.3 常见问题

表 12-3 列出了使用 COM 生成器时可能遇到的一些问题，以及引起问题的原因和解决问题的办法。

表 12-3 使用 COM 生成器引起的问题和解决办法

出 错 信 息	引起问题的原因	解决问题的办法
MBUILD.BAT:Error:The chosen compiler does not support building COM objects.	所选择的编译器不支持生成 COM 对象	重新运行 mbuild –setup，并选择一种编译器
Error in component_name.class_name.1_0:Error getting data conversion flags.	一般是因为没有注册 mwcomutil.dll	打开一个 DOS 窗口，将目录改变为 <matlabroot>\bin\win32，并运行命令 mwregsvr mwcomutil.dll

续表

出 错 信 息	引起问题的原因	解决问题的办法
Error in VBAProject:ActiveX component can't create object.	1.没有注册工程 DLL 2.在系统路径上的某个地方存在不兼容的 MATLAB DLL	如果 DLL 没有注册，打开一个 DOS 窗口，将路径改变为<projectdir>\distrib,<projectdir>表示工程文件的位置，并运行下面的命令：mwregsvr <projectdll>.dll
Error in VBAProject:Automation error.The specified module could not be found.	MATLAB 没有位于系统路径上	将<matlabroot>\bin\win32 放在路径上
LoadLibrary("component_name_1_0.dll") failed-The specified module could not be found.	从 DOS 提示符中注册工程 DLL 时，MATLAB 不在系统路径上	将<matlabroot>\bin\win32 放在路径上
Cannot recompile the M file xxxx because it is already in the library libmmfile.mlib.	所选择的 M 文件的名称与库中 M 文件的名称相同	重命名 M 文件，选择一个不与库中 M 文件冲突的名称

12.5 深入 COM 生成器组件

本节讨论与 COM 生成器组件有关的比较深入的话题，如版本、注册、数据类型转换等。深入了解这些知识对于更好地理解和使用 COM 组件十分有益。

12.5.1 COM 组件的兼容性

MATLAB 的 COM 生成器使得可以将经过编译的 MATLAB 对象集成到 Visual Basic, C++或任何其他支持 COM 的语言中。COM 生成器组件作为独立的 COM 对象生成。组件中的 MATLAB 函数作为 COM 对象的方法出现。

COM 生成器提供了健壮的数据转换机制和数组格式化，以保持从 Visual Basic 调用时 MATLAB 的灵活性。另外还提供了对错误的处理，这样，源于 MATLAB 函数的错误将自动作为 Visual Basic 的错误进行处理，返回的出错信息总是引用原来的 MATLAB 代码，使得调试更加容易。

另外，每个组件还创建了简单的版本机制，以帮助管理同一组件不同版本的配置。

12.5.2 组件生成的内部过程

创建 COM 组件的过程是完全自动进行的。用户提供了一个要处理的 M 文件列表和少量附加信息，如组件名、类名和版本号等。后面的创建工程包括代码生成、编译、链接和注册最终的组件。

（1）第 1 步生成创建组件的所有源代码和其他支持文件。编译器首先创建.c 和.h 文件（foo.h, foo.c, bar.h 和 bar.c），可以看成是原来 M 文件中代码的 C 语言翻译。它还创建包含每个 DLL 输出函数实现的主源文件（mycomponent_dll.cpp ）。编译器另外生成一个接口描述语言（IDL）文件（mycomponent_idl.idl），包括用相关 GUID 指定的组件的类型库、接口和类。GUID 是 Globally Unique Identifier（全局惟一标识符）的缩写，一个 128 位的整数值保证它总是惟一的。

后面创建的是 C++类的定义和实现文件(mycomponent_com.hpp 和 mycomponent_com.cpp)，除了这些源文件，编译器还生成一个 DLL 输出文件（mycomponent.def）和一个源脚本

（mycomponent.rc）。

（2）第 2 步是对第 1 步生成的 IDL 文件（mycomponent_idl.idl）调用 IDL 编译器，创建接口头文件（mycomponent_idl.h）、接口 GUID 文件（mycomponent_idl_i.c）和组件类型库文件（mycomponent_idl.tlb）。接口头文件包括 IDL 文件中基于接口定义的类型定义和函数声明。接口 GUID 文件包括 IDL 文件中源于所有接口的 GUID 的定义。组件类型库文件包括所有类型的二进制表示和组件提供的对象。

（3）第 3 步把前两步生成的所有 C/C++源代码编译为目标代码。这里有一个包含一系列 C++模板类（mclcomclass.h）的其他文件。本文件包括必要的 COM 基类、误差控制和注册代码的模板实现。

（4）第 4 步为组件生成最终的 DLL。本步调用第 3 步生成的对象文件的链接和必要的 MATLAB 库以生成 DLL 组件（mycomponent_1.0.dll）。然后源编译器在 DLL 上与第 1 步生成的源脚本一起被调用，以将第 1 步生成的类型库绑定到完成的 DLL 中。

（5）最后一步是在系统上注册 DLL。

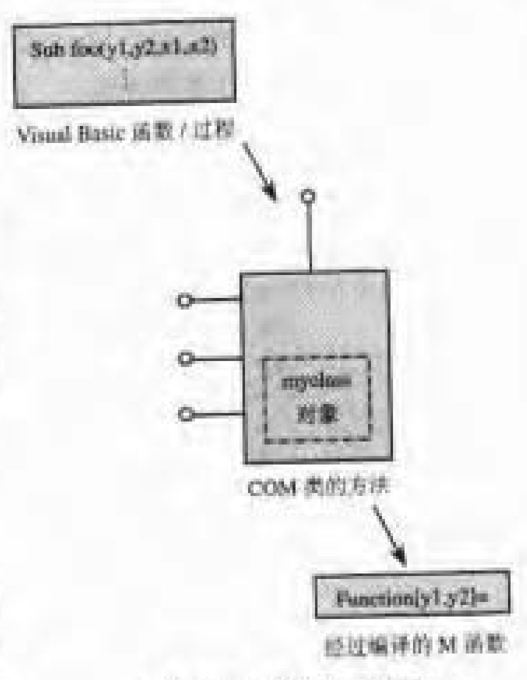


图 12-9　函数调用过程

12.5.3　调用约定

本节介绍 MATLAB COM 生成器组件的调用约定，包括从原始 M 文件向 Visual Basic 映射。Visual Basic 函数调用经过编译的 M 函数的机制如图 12-9 所示。

1．生成 COM 类

生成 COM 类需要生成用接口描述语言（IDL）编写的类定义文件，以及相关的 C++类定义/实现文件。生成器会自动生成必要的 IDL 和 C/C++代码来生成组件的每个 COM 类。这个过程对于用户来说，通常是透明的。

2．IDL 映射

MATLAB M 函数的一般形式为：

```
function [Y1, Y2, ..., varargout] = foo(X1, X2, ..., varargin)
```

该函数直接映射为下面的 IDL 代码：

```
HRESULT foo([in] long nargout,
            [in,out] VARIANT* Y1,
            [in,out] VARIANT* Y2,
            .
            .
            [in,out] VARIANT* varargout,
            [in] VARIANT X1,
            [in] VARIANT X2,
            .
            .
 [in] VARIANT varargin);
```

该 IDL 函数的名称与原 M 函数的相同，并且有一个包含原函数所有输入/输出参数加上 nargout 参数的参数列表。如果编译一个没有输出的 M 函数，将不生成 nargout 参数。如果该参数存在，则它是一个 long 型的[in]参数。它总是列表的第 1 个参数。使用该参数，可以正确地将 MATLAB 的 nargout 参数过渡为编译后的 M 代码。

nargout 参数后面，输出按参数在 MATLAB 函数左端出现的次序列出，并且用[in,out]标出，表示它们在两个方向上都可以传递。然后列出输入参数，按它们在 MATLAB 函数右端出现的次序列出。所有输入参数用[in]标出。如果存在可选的 varargin/varargout 参数，它们总是显示在输入参数和输出参数的最后。所有除 nargout 以外的参数都作为 COM VARIANT 类型进行传递。

3. Visual Basic 映射

上面的 foo 函数可以映射为下面的 Visual Basic 代码：

```
Sub foo(nargout As Long, _
        Y1 As Variant, _
        Y2 As Variant, _
        .
        .
        varargout As Variant, _
        X1 As Variant, _
        X2 As Varaint, _
        .
        .
        varargin As Variant)
```

Visual Basic 用 Variant 类型为 COM 的 VARIANT 类型，以及所有 Visual Basic 基本类型与 Variant 类型的转换提供了本地支持。通常，Visual Basic 基本类型的数组/标量和 Variant 类型的数组/标量可以作为参数传递。COM 生成器组件还提供了对 Excel 的 Range 对象的直接支持，该对象被 VBA 用于表示 Excel 工作表中的单元范围。

12.5.4 组件注册

COM 生成器创建组件时，会自动生成一个称为类型库的二进制文件。作为创建过程的最后一步，该文件作为资源文件与生成的 DLL 绑定在一起。

1. 自注册的组件

所有 COM 生成器组件都是自注册的。自注册组件包含向系统注册表添加或从系统注册表删除其完整描述的所有必要代码。使用 mwregsvr 工具注册自注册 DLL。例如，要注册一个名为 mycomponent_1_0.dll 的组件，在 DOS 命令行发出下面的命令：

```
mwregsvr mycomponent_1_0.dll
```

mwregsvr 完成注册过程以后，会显示一则表示成功或失败的消息。类似地，下面的命令：

```
mwregsvr /u mycomponent_1_0.dll
```

注销该组件。

将 COM 生成器组件安装到特定机器上，必须用 mwregsvr 工具进行注册。如果将组件移到同一机器的不同目录下，必须重新进行注册。从特定机器上删除组件时，首先注销它，以确保注册表中不剩下错误信息。

2. 全球惟一标识符

信息用一个或多个相关命名值作为密匙保存在注册表中。密匙本身主要具有两种类型的值，即可读字符串和 GUID。GUID 用一个 128 位的整数来确保其惟一性。MATLAB 编译器会自动生成创建组件时在组件内部定义的 COM 类、接口和类型库的 GUID 值，并且将这些密匙编码成组件的自注册码。至系统注册表的接口基于目录，并且与 COM 有关的信息保存在名为 HKEY_CLASSES_ROOT 的顶级密匙下。HKEY_CLASSES_ROOT 下是几个其他密匙，组件在这些密匙下写入信息。这些密匙如表 12-4 中所示。

表 12-4 密匙及其定义

密 匙	定 义
HKEY_CLASSES_ROOT\CLSID	系统上与 COM 有关的信息。每个组件在 HKEY_CLASSES_ROOT\CLSID 下为它的每个 COM 类创建一个新密匙。创建的密匙具有一个已经赋给类的 GUID 值，并且包含几个子密匙，它们具有与类有关的信息
HKEY_CLASSES_ROOT\Interface	系统上与 COM 接口有关的信息。每个组件在 HKEY_CLASSES_ROOT\Interface 目录下为它定义的每个接口创建一个新密匙。该密匙具有指定给接口的 GUID 值，并包含一些子密匙，它们具有与接口有关的信息
HKEY_CLASSES_ROOT\TypeLib	系统上与类型库有关的信息。每个组件通过将 GUID 值指定给类型库来为它创建一个密匙。在该密匙下，为类型库的每个版本创建一个新密匙。所以，具有相同名称的类型库的新版本重新使用原 GUID，但是创建一个新的子密匙
HKEY_CLASSES_ROOT\<ProgID>, HKEY_CLASSES_ROOT\<VerIndProgID>	这两个密匙是为组件的程序 ID 和版本独立程序 ID 创建的，用 <component-name>.<class-name>和<component-name>.<class-name><version-number>形式的字符串创建。根据组件和类的名称而不是 GUID 值创建类实例时，这些密匙很有用

3. 获取注册信息

COM 生成器包含了 MATLAB 函数 componentinfo，可以在系统注册表中查询安装的任何 COM 生成器组件。该函数可以在 MATLAB 内部执行，输入参数为组件名、主版本号和次版本号。它返回一个包含必要信息的结构数组。调用不带参数的 componentinfo 函数会返回安装在机器上的所有 COM 生成器组件。

下面的例子在注册表中查询一个组件名为 mycomponent、版本号为 1.0 的组件。该组件有 4 个方法，即 mysum,randvectors,getdates 和 myprimes；2 个属性，即 m 和 n；1 个事件，即 myevent。

```
Info = componentinfo('mycomponent', 1, 0)
Info =
Name: 'mycomponent'
        TypeLib: 'mycomponent 1.0 Type Library'
LIBID: '{3A14AB34-44BE-11D5-B155-00D0B7BA7544}'
MajorRev: 1
MinorRev: 0
FileName: 'D:\Work\ mycomponent\distrib\mycomponent_1_0.dll'
Interfaces: [1x1 struct]
CoClasses: [1x1 struct]

Info.Interfaces
ans =
        Name: 'Imyclass'
```

```
IID: '{3A14AB36-44BE-11D5-B155-00D0B7BA7544}'

Info.CoClasses
ans =
Name: 'myclass'
CLSID: '{3A14AB35-44BE-11D5-B155-00D0B7BA7544}'
ProgID: 'mycomponent.myclass.1_0'
VerIndProgID: 'mycomponent.myclass'
InprocServer32:'D:\Work\mycomponent\distrib\mycomponent_1_0.dll'
Methods: [1x4 struct]
Properties: {'m', 'n'}
Events: [1x1 struct]

Info.CoClasses.Events.M
ans =
function myevent(x, y)

Info.CoClasses.Methods
ans =
1x4 struct array with fields:
    IDL
    M
    C
    VB

Info.CoClasses.Methods.M
ans =
function [y] = mysum(varargin)
ans =
function [varargout] = randvectors()
ans =
function [x] = getdates(n, inc)
ans =
function [p] = myprimes(n)
```

返回的结构包含对应于组件注册表和类型库最重要信息的字段。这些字段在表 12-5 中定义。

表 12-5 组件返回的注册表信息

字 段	描 述
Name	组件名
TypeLib	组件类型库
LIBID	组件类型库 GUID
MajorRev	主版本号
MinorRev	次版本号
FileName	类型库文件名和路径，因为所有 COM 生成器组件都将类型库绑定到 DLL 中了，该文件名与 DLL 名称和路径相同
Interfaces	定义类型库中所有接口的结构数组。每个结构包含 2 个字段： • Name 接口名 • IID 接口 GUID

续表

字　段	描　述
CoClasses	定义组件中所有 COM 类的结构数组。每个结构包含下面这些字段： • Name　类名 • CLSID　类的 GUID • ProgID　版本依赖程序 ID • VerIndProgID　版本独立程序 ID • InprocServer32　组件 DLL 的完整名称和路径 • Methods　包含定义该接口的所有类方法的函数原型的结构。该结构包含 4 个字段： ➢ IDL　接口描述语言函数原型组成的数组 ➢ M　MATLAB 函数原型组成的数组 ➢ C　C 语言函数原型组成的数组 ➢ VB　Visual Basic 函数原型组成的数组 • Properties　包含所有类属性名称的单元数组 • Events　包含定义该类所有事件的函数原型的结构。该结构包含 4 个字段： ➢ IDL　接口描述语言函数原型组成的数组 ➢ M　MATLAB 函数原型组成的数组 ➢ C　C 语言函数原型组成的数组 ➢ VB　Visual Basic 函数原型组成的数组

12.5.5 版本控制

COM 生成器组件支持简单的版本机制，该机制使得生成和部署同一组件的多个版本很容易实现。组件的版本号是 DLL 名称的一部分，也是系统注册表中版本依赖 ID 的一部分。

创建组件以后，可以指定一个版本号，默认时为 1.0。开发组件的特定版本时，版本号应该保持不变。实现这一点以后，在特定条件下，MATLAB 编译器会在随后每次生成组件时重复使用类型库、类和接口 GUID。这避免了多次生成同一组件时创建过多的注册密匙，如果每次生成组件时都生成新的 GUID，就会发生这种情况。

得到新的版本号以后，MATLAB 编译器会生成新的类和接口 GUID，使得即使在类名相同的情况下，系统也能将它们与以前的版本区分开来。所以，部署已经生成的组件时，给组件的每次改变使用新的版本号。这确保了部署新组件以后可以比较容易地管理这两个版本。

MATLAB 编译器通过在系统注册表中查询已经存在的具有相同名称的组件来实现特定组件名、类名和版本号的版本控制。具体内容为：

• 如果已有组件具有相同的版本，它使用该组件类型库的 GUID 值。如果新类的名称与前一版本相匹配，则重新使用类和接口 GUID。如果类名不匹配，会给新类和接口生成新的 GUID。

• 如果查找一个具有不同版本的已有组件，使用已有类型库的 GUID，并为新版本号创建新的子密匙。它给新类和接口生成新的 GUID。

• 如果不查找指定名称的已有组件，会给组件类型库、类和接口生成新的 GUID。

12.5.6 数据转换

CGM 生成器组件是双重接口的 CGM 对象，它支持 COM 自动化兼容的数据类型。调用 COM 生成器组件的方法时，输入参数将转换为 MATLAB 内部数组格式并传递给经过编译的 MATLAB 函数。函数退出时，输出参数从 MATLAB 内部数组格式转换为 COM 自动

化类型。

COM 客户端在经过编译的 MATLAB 函数中将所有的输入、输出变量作为 VARIANT 型变量进行传递。一个 VARIANT 类型的变量可以保存任何简单类型的变量。Win32 应用程序接口（API）提供了许多创建和操作 VARIANT 类型变量的函数，Visual Basic 提供了支持这种类型的本地语言。

表 12-6 列出了 COM 生成器组件支持的 VARIANT 类型码。

表 12-6　COM 生成器组件支持的 VARIANT 类型码

VARIANT 类型码 (C/C++)	C/C++类型	Variant 类型码 (Visual Basic)	Visual Basic 类型	定　义
VT_EMPTY	-	VbEmpty	-	没有初始化的 VARIANT
VT_I1	char	-	-	有符号单字节字符
VT_UI1	unsigned char	vbByte	Byte	无符号单字节字符
VT_I2	short	vbInteger	Integer	有符号双字节整数
VT_UI2	unsigned short	-	-	无符号双字节整数
VT_I4	long	vbLong	Long	有符号 4 字节整数
VT_UI4	unsigned long	-	-	无符号 4 字节整数
VT_R4	float	vbSingle	Single	IEEE 4 字节浮点值
VT_R8	double	vbDouble	Double	IEEE 8 字节浮点值
VT_CY	CY*	vbCurrency	Currency	货币值(64 位整数、用 10,000 进行比例化)
VT_BSTR	BSTR*	vbString	String	字符串值
VT_ERROR	SCODE*	vbError	-	HRESULT (表示 COM 出错码的有符号 4 字节整数)
VT_DATE	DATE*	vbDate	Date	表示日期和时间的 8 字节浮点值
VT_INT	int	-	-	有符号整数，等价于 int 类型
VT_UINT	unsigned int	-	-	无符号整数，等价于无符号 int 型
VT_DECIMAL	DECIMAL*	vbDecimal	-	96 位(12 字节) 无符号整数，用 10 的可变次幂进行比例化
VT_BOOL	VARIANT_BOOL*	vbBoolean	Boolean	2 字节布尔型值(0xFFFF = True; 0x0000 = False)
VT_DISPATCH	IDispatch*	vbObject	object	IDispatch* 指向对象的指针
VT_VARIANT	VARIANT*	vbVariant	Variant	VARIANT（只能在与 VT_BYREF 或 VT_ARRAY 进行组合时指定)
<anything> | VT_ARRAY				用声明为数组的任何基本类型逐位组合 VT_ARRAY
<anything> | VT_BYREF				用声明为引用的任何基本类型逐位组合 VT_BYREF

* 表示 Windows 特定类型，不是标准 C/C++ 的一部分。

表 12-7 和表 12-8 列出了 COM 的 VARIANT 型和 MATLAB 数组之间的转换规则。

表 12-7　MATLAB 向 COM VARIANT 转化的规则

MATLAB 数据类型	标量数据的 VARIANT 类型	数组数据的 VARIANT 类型	注　释
cell	1×1 的单元数组转化为单一的 VARIANT，类型为单元内容的 MATLAB 数据类型遵守转换规则进行转换以后得到的类型	多维单元数组转换为 VT_VARIANT|VT_ARRAY 类型的 VARIANT，每个数组元素的类型是对应单元的 MATLAB 数据类型按照转换规则进行转换以后得到的类型	
structure	VT_DISPATCH	VT_DISPATCH	MATLAB 结构数组转换为 MWStruct 对象，该对象作为 VT_DISPATCH 类型进行传递
char	1×1 的字符矩阵转换为 VT_BSTR 类型的 VARIANT，字符串长度等于 1	MATLAB 中，1×L 的字符矩阵表示长度为 L 的字符串。它转换为 VT_BSTR 类型的 VARIANT，长度为 L。一行以上，或者具有更高维的字符矩阵转换为 VT_BSTR|VT_ARRAY 类型的 VARIANT。转换后的数组中，每个字符串的长度为 1，对应于原矩阵中的每个字符	字符串数组不作为字符矩阵进行支持。使用 1×L 的字符矩阵单元数组传递字符串单元数组
sparse	VT_DISPATCH	VT_DISPATCH	MATLAB 稀疏数组转换为 MWSparse 对象，该对象作为 VT_DISPATCH 类型进行传递
double	实数型的 1×1 double 型矩阵转换为 VT_R8 类型的 VARIANT。复数型的 1×1 double 型矩阵转换为 VT_DISPATCH 类型的 VARIANT	实数型的多维 double 型矩阵转换为 VT_R8|VT_ARRAY 类型的 VARIANT。复数型的多维 double 型矩阵转换为 VT_DISPATCH 类型的 VARIANT	复数数组用 MWComplex 类传入经过编译的 M 函数，或从该函数传出
single	实数型的 1×1 single 型矩阵转换为 VT_R4 类型的 VARIANT。复数型的 1×1 single 型矩阵转换为 VT_DISPATCH 类型的 VARIANTA	实数型的多维 single 型矩阵转换为 VT_R4|VT_ARRAY 类型的 VARIANT。复数型的多维 single 型矩阵转换为 VT_DISPATCH 类型的 VARIANTA	复数数组用 MWComplex 类传入经过编译的 M 函数，或从该函数传出
int8	实数型的 1×1 int8 型矩阵转换为 VT_I1 类型的 VARIANT。复数型的 1×1 int8 型矩阵转换为 VT_DISPATCH 类型的 VARIANTA	实数型的多维 int8 型矩阵转换为 VT_I1|VT_ARRAY 类型的 VARIANT。复数型的多维 int8 型矩阵转换为 VT_DISPATCH 类型的 VARIANTA	复数数组用 MWComplex 类传入经过编译的 M 函数，或从该函数传出
uint8	实数型的 1×1 uint8 型矩阵转换为 VT_UI1 类型的 VARIANT。复数型的 1×1 uint8 型矩阵转换为 VT_DISPATCH 类型的 VARIANTA	实数型的多维 uint8 型矩阵转换为 VT_UI1|VT_ARRAY 类型的 VARIANT。复数型的多维 uint8 型矩阵转换为 VT_DISPATCH 类型的 VARIANTA	复数数组用 MWComplex 类传入经过编译的 M 函数，或从该函数传出
int16	实数型的 1×1 int16 型矩阵转换为 VT_I2 类型的 VARIANT。复数型的 1×1 int16 型矩阵转换为 VT_DISPATCH 类型的 VARIANTA	实数型的多维 int16 型矩阵转换为 VT_I2|VT_ARRAY 类型的 VARIANT。复数型的多维 int16 型矩阵转换为 VT_DISPATCH 类型的 VARIANTA	复数数组用 MWComplex 类传入经过编译的 M 函数，或从该函数传出
uint16	实数型的 1×1 uint16 型矩阵转换为 VT_UI2 类型的 VARIANT。复数型的 1×1 uint16 型矩阵转换为 VT_DISPATCH 类型的 VARIANTA	实数型的多维 uint16 型矩阵转换为 VT_UI2|VT_ARRAY 类型的 VARIANT。复数型的多维 uint16 型矩阵转换为 VT_DISPATCH 类型的 VARIANTA	复数数组用 MWComplex 类传入经过编译的 M 函数，或从该函数传出

续表

MATLAB 数据类型	标量数据的 VARIANT 类型	数组数据的 VARIANT 类型	注　释
int32	实数型的 1×1 int32 型矩阵转换为 VT_I4 类型的 VARIANT。复数型的 1×1 int32 型矩阵转换为 VT_DISPATCH 类型的 VARIANTA	实数型的多维 int32 型矩阵转换为 VT_I4|VT_ARRAY 类型的 VARIANT。复数型的多维 int32 型矩阵转换为 VT_DISPATCH 类型的 VARIANT	复数数组用 MWComplex 类传入经过编译的 M 函数，或从该函数传出
uint32	实数型的 1×1 uint32 型矩阵转换为 VT_UI4 类型的 VARIANT。复数型的 1×1 uint32 型矩阵转换为 VT_DISPATCH 类型的 VARIANTA	实数型的多维 uint32 型矩阵转换为 VT_UI4|VT_ARRAY 类型的 VARIANT。复数型的多维 uint32 型矩阵转换为 VT_DISPATCH 类型的 VARIANT	复数数组用 MWComplex 类传入经过编译的 M 函数，或从该函数传出
函数句柄	VT_EMPTY	VT_EMPTY	不支持
Java 类	VT_EMPTY	VT_EMPTY	不支持
用户类	VT_EMPTY	VT_EMPTY	不支持
逻辑型	VT_Bool	VT_Bool|VT_ARRAY	

表 12-8　COM VARIANT 向 MATLAB 的转换规则

VARIANT 类型	MATLAB 数据类型（标量或数组数据）	说　明
VT_EMPTY	N/A	创建空数组
VT_I1	int8	
VT_UI1	uint8	
VT_I2	int16	
VT_UI2	uint16	
VT_I4	int32	
VT_UI4	uint32	
VT_R4	single	
VT_R8	double	
VT_CY	double	
VT_BSTR	char	VT_BSTR 类型的 VARIANT 转换为 1×L 的 MATLAB 字符数组，其中 L 为要转换的字符串的长度。VT_BSTR·VT_ARRAY 类型的 VARIANT 转换为 1×L 字符数组构成的 MATLAB 单元数组
VT_BRROR	int32	
VT_DATE	double	1．VARIANT 日期作为 double 型数据保存，起始日期为 1899 年 12 月 31 日晚 12 时。MATLAB 日期的起始时间是 0/0/00 00:00:00。所以，VARIANT 日期 0.0 映射到 MATLAB 数值日期 693960.0，VARIANT 日期转换为 MATLAB double 类型，增加了 693960.0 2．VARIANT 日期可以可选地转换为字符串
VT_INT	int32	
VT_UINT	uint32	
VT_DECIMAL	double	
VT_BOOL	logical	
VT_DISPATCH	(变化)	IDispatch*指针类型根据它们指向对象所处的上下文来确定
<anything>|VT_BYREF	(变化)	任何基本类型的指针根据它们指向的对象的类型进行处理。生成的 MATLAB 数组包含值的深拷贝
<anything>|VT_ARRAY	(变化)	多维 VARIANT 数组转换为多维 MATLAB 数组，每个元素根据基本类型进行转换。VT_VARIANT|VT_ARRAY 类型的的多维 VARIANT 数组转换为多维单元数组，每个单元根据特定类型进行转换

1. 数组格式化标记

COM 生成器组件具有控制数组数据如何在两个方向上进行格式化的标记。通常应该开发匹配 MATLAB 函数和编译后 COM 对象的对应方法的输入和输出参数的客户代码，按照表 12-7 和表 12-8 所示的规则进行匹配。

表 12-9 显示了数组格式化标记。

表 12-9 数组格式化标记

标　记	描　述
InputArrayFormat	定义用于输入数组的数组格式化规则。输入数组是 VARIANT 数组，由客户创建，作为输入参数发送给编译后 COM 对象的方法。该标记的合法值包括 mwArrayFormatAsIs,mwArrayFormatMatrix 和 mwArrayFormatCell。 mwArrayFormatAsIs 传递没改变的数组。 mwArrayFormatMatrix 为默认值，将所有数组作为矩阵处理。当输入 VARIANT 是 VT_ARRAY | <type> 类型的时(其中<type>为任何数值类型)，标记没有影响。当输入 VARIANT 是 VT_VARIANT VT_ARRAY 类型的时，探察数组中的 VARIANT。如果它们是单值的并且类型相同，则生成一个对应类型的 MATLAB 矩阵，而不是单元数组。 mwArrayFormatCell 将所有数组作为 MATLAB 单元数组进行解释。
InputArrayIndFlag	设置输入数组的间接级别，它与 InputArrayFormat 标记一起使用。是适用于嵌套数组，即 VARIANT 数组，其元素也是数组。该标记的默认值为 0，它将 InputArrayFormat 标记用于最外层数组。当标记大于 0，例如等于 N 时，格式化规则试图将它本身应用于嵌套的第 N 个级别
OutputArrayFormat	定义用于输出数组的数组格式化规则。输出数组是一个 MATLAB 数组，由编译后的 COM 对象创建，作为输出参数从调用的方法发送给客户，该标记的值可以是 mwArrayFormatAsIs,,mwArrayFormatMatrix 和 mwArrayFormatCell，与 InputArrayFormat 标记的值相对应
OutputArrayIndFlag	该标记只用于嵌套的单元数组，设置输出数组的间接级别，与 OutputArrayFormat 标记一起使用。工作方式与 InputArrayIndFlag 标记的完全相同
AutoResizeOutput	只用于 Excel 的范围。当调用方法的目标输出是 Excel 工作表中的单元范围，并且调用时输出数组的大小和形状未知，将该标记的值设置为 True 来改变 Excel 电子表格范围的大小，使它适合输出数组的大小
TransposeOutput	将该标记设置为 True，转置输出参数。从 Excel 中调用 COM 生成器组件时，如果 MATLAB 函数将输出作为行矢量返回，但希望数据按列排列时该标记很有用

2. 数据转换标记

COM 生成器组件包含控制将特定 VARIANT 类型转换为 MATLAB 类型的标记。这些标记包括：

（1）CoerceNumericToType 标记

该标记让数据转换器将所有数值型 VARIANT 数据转换为一种特定的 MATLAB 类型。本标记可以影响的 VARIANT 类型码包括 VT_I1,VT_UI1,VT_I2,VT_UI2,VT_I4,VT_UI4,VT_R4, VT_R8,VT_CY,VT_DECIMAL,VT_INT,VT_UINT,VT_ERROR,VT_BOOL 和 VT_DATA。该标记的合法值包括 mwTypeDefault,mwTypeChar,mwTypeDouble,mwTypeSingle,mwTypeLogical, mwTypeInt8,mwTypeUint8,mwTypeInt16,mwTypeUint16,mwTypeInt32 和 mwTypeUint32。

（2）InputDateFormat 标记

本标记告诉数据转换器如何将 VARIANT 日期转换为 MATLAB 日期。该标记的合法值为 mwDataFormatNumeric（默认）和 mwDateFormatString。

（3）OutputAsDate 标记

为布尔型。该标记让数据转换器将输出参数作为日期处理。默认时，作为经过编译的

MATLAB 函数的输出参数的数值型日期作为 Double 型值进行传递，需要减去 COM 日期偏移(693960)。将该标记设置为 True 时将所有输出值的类型都转换为 Double 型。

（4）DateBias 标记

为 Long 型。该标记设置进行 COM 至 MATLAB 数值型日期转换时的日期偏移。该属性的默认值是 693960，它表示 COM Date 类型和 MATLAB 数值型日期之间的差值。

12.6 工具库

本节介绍与 MATLAB Excel 生成器一起提供的 MWComUtil 库。本库自由发布，包括几个用于数组处理的函数和用于数据转换的类型定义。本库包含在文件 mwcomutil.dll 中。它必须在使用生成器组件的机器上注册一次。

用下面的 DOS 命令在命令提示行中注册 MWComUtil 库.

```
mwregsvr mwcomutil.dll
```

MWComUtil 库包括 7 个类和 3 个枚举类型。在使用这些类之前，必须在 VB IDE 中引用 MWComUtil 类库。要做到这些，在 Visual Basic 编辑器的主菜单中依次选择 Tools→References…。“References”对话框中列出了所有可以获得的类型库。从该列表中选择“MWComUtil 1.0 Type Library”，并单击“OK”按钮。

Excel 生成器工具库提供了以下几个类：

- MWUtil 类；
- MWFlags 类；
- MWStruct 类
- MWField 类；
- MWComplex 类；
- MWSparse 类；
- MWArg 类。

12.6.1 MWUtil 类

MWUtil 类包含一系列用于数组处理和应用初始化的静态工具方法。使用本类时，每个 Excel 实例中只能有一个本类的全局实例。使用该全局实例时，最好把它定义为全局变量。下面介绍 MWUtil 类的方法过程。

1. MWInitApplication(pApp As Object)过程

该过程用当前的 Excel 实例初始化库。其中，pApp 参数为 Object 类型，是当前 Excel 应用的一个合法引用，无返回值。

本过程必须被每个使用 Excel 生成器组件的 MATLAB 进程调用一次。如果库没有初始化，则调用该库的成员类时将产生错误。

【例 1】 本范例用当前的 Excel 实例初始化 MWComUtil 库，一个 Object 类型的全局变量 MCLUtil 包含一个 MWUtil 类，另一个 Boolean 型的全局变量 bModuleInitialized 保存初始化过程的状态。私有过程 InitModule 创建 MWComUtil 类的一个实例，并调用变量 Application 的 MWInitApplication 方法。一旦本函数成功，所有后面的调用将在没有创建对象的情况下退出。

```
Dim MCLUtil As Object
Dim bModuleInitialized As Boolean

Private Sub InitModule()
    If Not bModuleInitialized Then
        On Error GoTo Handle_Error
        If MCLUtil Is Nothing Then
            Set MCLUtil = CreateObject("MWComUtil.MWUtil")
        End If
        Call MCLUtil.MWInitApplication(Application)
        bModuleInitialized = True
        Exit Sub
Handle_Error:
        bModuleInitialized = False
    End If
End Sub
```

2. MWPack(pVarArg,[Var0],[Var1],…,[Var31])过程

该过程把一个 Variant 型的变量长度列表打包到单一的 Variant 型数组中。本过程的典型应用是在有多个输入时创建一个包含这些输入的 varargin 单元。列表中的输入非空或非缺失值时，被添加到数组中去（在 VB 中，一个缺失数据的参数用值为&H80020004 的 Variant 型的 vbError 表示）。

参数 pVarArg 为 Variant 型，获取生成的数组；[Var0]、[Var1]等参数也是 Variant 型的，是数组中的 Variant 型数据的可选列表。

注意：在处理列表前，本函数总是释放 pVarArg 的内容。

【例 2】 本例使用公式函数中的 MWPack 函数创建一个 varargin 单元，作为一个输入参数传递到一个方法。该方法从一个 MATLAB 函数编译而来。

```
function y = mysum(varargin)
    y = sum([varargin{:}]);
```

该函数返回 varargin 中元素的和。假设本函数是一个 myclass 类的方法，myclass 类被包含在一个名为 mycomonent 的组件中，版本为 1.0。VB 函数最多允许有 10 个输入，返回结果 y。如果发生错误，该函数返回错误字符。本函数假设前面已经调用了 MWInitApplication 方法。

```
Function mysum(Optional V0 As Variant, _
    Optional V1 As Variant, _
    Optional V2 As Variant, _
    Optional V3 As Variant, _
    Optional V4 As Variant, _
    Optional V5 As Variant, _
    Optional V6 As Variant, _
    Optional V7 As Variant, _
    Optional V8 As Variant, _
    Optional V9 As Variant) As Variant
    Dim y As Variant
    Dim varargin As Variant
    Dim aClass As Object
    Dim aUtil As Object
```

```
        On Error Goto Handle_Error
        Set aClass = CreateObject("mycomponent.myclass.1_0")
        Set aUtil = CreateObject("MWComUtil.MWUtil")
        Call aUtil.MWPack(varargin,V0,V1,V2,V3,V4,V5,V6,V7,V8,V9)
        Call aClass.mysum(1, y, varargin)
        mysum = y
        Exit Function
    Handle_Error:
        mysum = Err.Description
    End Function
```

3. MWUnPack()过程

把 Variant 型数组解包到单一的 Vairant 型变量中，本函数提供了 MWPack 函数的反函数，常用于将 varargout 单元处理为多个单独的 Variant。

参数 varArg 为 Variant 型，是要处理的 Variant 型输入数组。nStartAt 为 Long 型，是开始处理时数组中的起始编号（从 0 开始），为可选项，默认值为 0。bAutoResize 参数为 Boolean 型，是自动改变大小的标记，可选。如果本标记为真，任何 Excel 的 Range 类型输出变量将改变大小，以适应要复制的 Variant 型数据的维。改变大小的过程相对于给定范围的左上角进行，默认时为 False。[pVar0], [pVar1]等参数为可选项，是 varArg 中数组选项的列表，可以传递 0 到 32 个变量。

注意：本过程可以用 nStartAt 参数，通过一次调用或多次调用来处理一个 Variant 型数组。

【例 3】 本例使用 MWUnpack 方法，在自动改变每个范围时，将一个 arargout 单元处理为多个 Excel 范围。varargout 参数由 MATLAB 函数编译得到的一个方法提供。

```
function varargout = randvectors
    for i=1:nargout
        varargout{i} = rand(i,1);
    end
```

本函数生成一系列 narargout 随机列矢量，第 i 个矢量的长度为 i。假设本函数位于版本 1.0 的 mycomponent 组件的 myclass 类中。Visual Basic 过程没有变量，把结果放入始于 A1, B1, C1 和 D1 的列中；如果发生错误，则显示出错文本信息框。本函数假设已经调用了 MWInitApplication 方法。

示例代码如下：

```
Sub GenVectors()
    Dim aClass As Object
    Dim aUtil As Object
    Dim v As Variant
    Dim R1 As Range
    Dim R2 As Range
    Dim R3 As Range
    Dim R4 As Range

    On Error GoTo Handle_Error
    Set aClass = CreateObject("mycomponent.myclass.1_0")
    Set aUtil = CreateObject("MWComUtil.MWUtil")
    Set R1 = Range("A1")
```

```
        Set R2 = Range("B1")
        Set R3 = Range("C1")
        Set R4 = Range("D1")
        Call aClass.randvectors(4, v)
        Call aUtil.MWUnpack(v,0,True,R1,R2,R3,R4)
        Exit Sub
    Handle_Error:
        MsgBox (Err.Description)
    End Sub
```

过程中将随机矢量数组 v 解包到 Excel 范围 R1、R2、R3 和 R4 中。

4. MWDate2VariantDate(pVar)过程

本过程将 MATLAB 的日期输出转换为 Variant 型日期。其中，pVar 为要转换的日期。

注意：MATLAB 将日期作为双精度浮点数处理，0.0 表示 0 /00/00 00:00:00。默认时，作为经过编译的输出参数的数值型日期，并作为 double 型数据传递。

【例 4】 本例使用 MWDate2VariantDate 过程处理数值型日期，该日期从经过编译的下面的 MATLAB 函数的方法中返回。

```
function x = getdates(n, inc)
    y = now;
    for i=1:n
        x(i,1) = y + (i-1)*inc;
end
```

本过程生成一个长度为 n 的数值型值的列矢量，这些值代表从当前日期和时间开始，然后每增加 inc 天得到一个新值后形成的一系列数据。假设本函数位于名为 mycomponent 的版本为 1.0 的组件的 myclass 类中，该过程将一个 Excel Range 对象和一个 double 型数据作为输入，并将生成的日期放在指定的范围中；如果发生错误，则用信息框显示出错文本。假设已经调用了 MWInitApplication 方法。

代码如下：

```
Sub GenDates(R As Range, inc As Double)
    Dim aClass As Object
    Dim aUtil As Object

    On Error GoTo Handle_Error
    Set aClass = CreateObject("mycomponent.myclass.1_0")
    Set aUtil = CreateObject("MWComUtil.MWUtil")
    Call aClass.getdates(1, R, R.Rows.Count, inc)
    Call aUtil.MWDate2VariantDate(R)
    Exit Sub
Handle_Error:
    MsgBox (Err.Description)
End Sub
```

12.6.2 MWFlags 类

MWFlags 类包含一系列数组格式化和数据转换标记。所有 MATLAB Excel 生成器组件都包含一个 MWFlags 对象的引用。它可以在对象的水平上修改数据转换的规则。本类包含下面

这些属性和方法：

（1）ArrayFormatFlags As MWArrayFormatFlags 属性。

（2）DateConversionFlags As MWDateConversinFlags 属性。

（3）Clone(ppFlags As MWFlags)方法。

1. ArrayFormatFlags 属性

本属性控制数组格式（作为矩阵还是单元数组），以及将这些规则应用于内嵌数组。MWArrayFormatFlags 类是一个不能通过 MWFlags 类实例创建的类，有以下 6 个属性。

（1）InputArrayFormat 属性。该属性控制数组的格式。这些数组作为输入参数传递给 MATLAB Excel 生成器的类方法，默认值为 mwArrayFormatMatrix，这些数组的格式化规范用表 12-10 列出的标记来表示。

表 12-10 输入数组的格式化规范

值	行　为
mwArrayFormatAsIs	根据默认的转换规则转换数组
mwArrayFormatMatrix	将所有数组强制转换为数组，当遇到的输入变量是一个 Variant 型数组时（默认时将它转换为单元数组），如果每个 Variant 为单一的值，数据转换器将本数组转换为一个矩阵，并且所有元素都是数值型的。如果此转换不可行，则创建一个单元数组
mwArrayFormatCell	将所有数组强制转换为单元数组，输入的标量或数值型数组转换为单元数组，每个单元包含一个标量值

（2）InputArrayIndFlag 属性。本属性控制应用由 InputArrayFormat 属性所设置的规则的水平。不必为 varargin 参数修改本属性。数据转换由代码自动给 varargin 单元标记值加 1，这样，将 InputArrayFormat 标记应用到 varargin 参数的每个单元，默认值为 0。

（3）OutputArrayFormat 属性。

（4）OutputArrayIndFlag 属性。

（5）AutoResizeOutput 属性。本属性只应用于 Excel 范围。当方法调用的目标输出是 Excel 工作表中一个范围内的单元，并且调用时输出数组的大小和形状不知道时，将此标记设为 True，指示数据转换代码，以改变每个 Excel 范围的大小，适应输出数组。改变大小时相对于每个指定范围的左上角进行，默认值为 False。

（6）TransposeOutput 属性。将这个属性设置为 True，对输出变量进行转置，当处理 Excel 生成器组件方法调用的输出参数时，本属性很有用，这里，MATLAB 函数将输出作为行矢量，并且需要将数据放到列中。本属性的默认值为 False。

2. DataConversionFlags 属性

该属性控制需要进行类型转换时，如何处理输入变量，它是 DataConversionFlags 类型的。MWDataConversionFlags 类不能通过 MWFlags 类的实例获取。本类包括下面的属性：

- CoerceNumericToType 属性；
- InputDateFormat 属性；
- OutputAsDate 属性；
- DateBias 属性。

（1）CoerceNumericToType 属性

本属性将所有数值型输入变量转换为一个指定的 MATLAB 类型。在 VB 代码中有多个不

同数据类型，且需要将它们转换为同一种数据类型的时候，这个标记很有用。该属性的默认值为 mwTypeDefault，它使用默认的转换规则。

（2）InputDateFormat 属性

本属性将作为输入参数的日期型数据转换为 Excel 生成器类的方法调用，默认值为 mwDateFormatNumeric。Date 数据的转换规则如表 12-11 所示。

表 12-11　Date 数据的转换规则

值	行　为
mwDateFormatNumeric	将日期型转为数值型
MwDateFormatString	将日期型转为字符串型

【例 5】 本例使用数据转换标记更新方法输出的形状。该方法从一个 MATLAB 函数编译以后得到，它返回一个长度未知的矢量。

```
function p = myprimes(n)
    if length(n)~=1, error('N must be a scalar'); end
    if n < 2, p = zeros(1,0); return, end
    p = 1:2:n;
    q = length(p);
    p(1) = 2;
    for k = 3:2:sqrt(n)
        if p((k+1)/2)
            p(((k*k+1)/2):k:q) = 0;
        end
    end
    p = (p(p>0));
```

本属性（函数）生成一个数值在 0 到 n 之间的行矢量，假设该函数位于版本为 1.0 的 mycomponent 组件的 myclass 类中。下面的过程将一个 Excel 范围和一个 Double 值作为输入，并将生成的数值放到指定的范围中，该函数生成一个行矢量。

```
Sub GenPrimes(R As Range, n As Double)
    Dim aClass As mycomponent.myclass

    On Error GoTo Handle_Error
    Set aClass = New mycomponent.myclass
    aClass.MWFlags.ArrayFormatFlags.AutoResizeOutput = True
    aClass.MWFlags.ArrayFormatFlags.TransposeOutput = True
    Call aClass.myprimes(1, R, n)
    Exit Sub
Handle_Error:
    MsgBox (Err.Description)
End Sub
```

（3）OutputAsDate 属性

本属性将一个输出变量处理为一个日期。默认时，把作为经过编译的 MATLAB 函数输出参数的数值型日期作为 Double 型值进行传递。将本标记设置为 True，将所有输出值转换为 Double 类型。

(4) DateBias 属性

本属性设置日期偏差，以进行 COM 到 MATLAB 数值型日期的转换。本属性的默认值为 693960，表示 COM Date 类型与 MATLAB 数值型日期之间的差异。

3. Clone 方法

本方法创建一个 MWFlags 对象的复制，参数 ppFlags 为 MWFlags 类型，是接收复制的未初始化的 MWFlags 对象的引用。

注意：Clone 方法分配一个新的 MWFlags 对象并创建对象内容的一个深度复制。当需要单个对象而不是一个已存在对象引用的共享复制时，调用本方法。

12.6.3 MWStruct 类

MWStruct 类与经过编译的类方法传递 struct 类型的数据，本类包含 7 个属性和方法。

1. Initialize 方法

本方法用一个指定的数值、维的大小和一个指定的域名列表分配一个结构数组。

参数 varDims 为维数组，可选；varFieldNames 为域名数值，可选。

注意：创建以后，有一个 1×1 的 MWStruct 对象，没有字段，Initialize 方法确定数组维数并添加一系列命名字段给每个元素。每次调用相同对象的 Initialize 方法时，都会重新定义维。如果不提供 varDims 变量，则不会改变已经存在的数和数组维的大小。如果不提供 varFieldNames 变量，则已经存在的字段列表不会改变，调用没有参数的 Initialize 方法同样不改变数组。

【例 6】 下面的 Visual Basic 代码演示 Initialize 方法的使用。

```
Sub foo ()
    Dim x As MWStruct
    Dim y As MWStruct

    On Error Goto Handle_Error
    '为 x 和 y 创建一个没有字段的 1×1 的结构数组
    Set x = new MWStruct
    Set y = new MWStruct

    '把 x 初始化为 2×2 的结构数组，字段包括 "red", "green", 和"blue"
    Call x.Initialize(Array(2,2), Array("red", "green", "blue"))
    '用 1×5 的结构数级初始化 y,字段为 "name" 和 "age"
    Call y.Initialize(5, Array("name", "age"))

    '把 x 的大小改为 3×3，字段名不变
    Call x.Initialize(Array(3,3))

    '给 y 添加一个新的字段
    Call y.Initialize(, Array("name", "age", "salary"))

    Exit Sub
Handle_Error:
    MsgBox(Err.Description)
End Sub
```

2. Item 属性

该属性是 MWStruct 类的默认属性。本属性用于设置 / 获取结构数组中一定编号的字段的值。

参数 i0,i1,…,i31 为可选项，可以是 0 到 31 之间的编号变量。若要引用数组的一个元素，则指定字段名的所有编号。

3. NumberOfFields 属性

该属性为只读属性。它返回结构数组中的字段个数。

4. NumberOfDims 属性

该属性为只读属性。它返回结构数组中的维数。

5. Dims 属性

该属性为只读属性。它返回包含结构数组中元素字段名称长度的数组。

【例 7】 下面的例子演示如何获取一个二维结构数组的字段。

```
Sub foo ()
    Dim x As MWStruct
    Dim Dims as Variant
    Dim FieldNames As Variant

    On Error Goto Handle_Error
    '... 调用一个返回 MWStruct 变量到 x 中的方法
    Dims = x.Dims
    FieldNames = x.FieldNames
    For I From 1 To Dims(1)
        For J From 1 To Dims(2)
            For K From 1 To x.NumberOfFields
                y = x(I,J,FieldNames(K))
                '... 用 y 做些什么
            Next
        Next
    Next
    Exit Sub
Handle_Error:
    MsgBox(Err.Description)
End Sub
```

6. Clone 方法

该方法创建一个 MWStruct 对象的复制。参数为 ppStruct 为 MWStruct 型，是一个接收复制的未初始化的 MWStruct 对象的引用。

【例 8】 本例演示赋值与 MWStruct 对象的 Clone 方法之间的不同。用已知对象 x1 创建新对象时，如果采用赋值方法得到新对象 x2，用 Clone 方法得到新对象 x3，则当 x1 改变时，x2 随之改变，x3 不变。

```
Sub foo ()
    Dim x1 As MWStruct
    Dim x2 As MWStruct
    Dim x3 As MWStruct
```

```
        On Error Goto Handle_Error
        Set x1 = new MWStruct
        x1("name") = "John Smith"
        x1("age") = 35
        '设置 x1 的引用为 x2
        Set x2 = x1
        '为 x3 创建新对象，并把 x1 的内容复制到 x3
        Call x1.Clone(x3)
        'x2 的 "age" 字段也改变了，'x3'的"age"字段不变
        x1("age") = 50
        Exit Sub
    Handle_Error:
        MsgBox(Err.Description)
    End Sub
```

7. FieldNames 属性

该属性为只读属性。它返回结构数组中元素字段名长度的数组。

12.6.4 MWField 类

该类包含的属性和方法如下。

1. Name 属性

该属性只读，字段名。

2. Value 属性

该属性为可读可写。它保存字段的值，是类的默认属性。

3. MWFlags 属性

该属性保存 MWFlags 对象的一个引用，为特定字段设置或获取数组格式化和数据转换标记。结构中的每个字段有它自己的属性。本属性重写被调用方法所属对象的所有标记。

4. Clone 方法

Clone 创建一个 MWField 对象的复制。

12.6.5 MWComplex 类

该类与经过编译的类方法传递或接收一个 Variant 型数组，本类包括 4 个属性和方法。

1. Real 属性

该属性保存复数数组的实数部分。它是 MWComplex 类的默认属性。

2. Imag 属性

Imag 保存复数数组的虚数部分，为可选项，对于纯实型数组而言可设为空。

下面的 VB 代码创建一个复数数组：

```
x = [ 1+i 1+2i
      2+i 2+2i ]
Sub foo()
    Dim x As MWComplex
    Dim rval(1 To 2, 1 To 2) As Double
```

```
        Dim ival(1 To 2, 1 To 2) As Double

        On Error Goto Handle_Error
        For I = 1 To 2
            For J = 1 To 2
                rval(I,J) = I
                ival(I,J) = J
            Next
        Next
        Set x = new MWComplex
        x.Real = rval
        x.Imag = ival

        Exit Sub
    Handle_Error:
        MsgBox(Err.Description)
    End Sub
```

3. MWFlags 属性

该属性保存一个 MWFlags 对象的引用。本属性设置或获取特定复数数组的数组格式化和数据转换标记。每个 MWComplex 对象有它自己的 MWFlags 属性。

4. CLone 方法

该方法创建 MWComplex 对象的一个复制。

12.6.6 MWSparse 类

该类与经过编译的类方法之间相互传递一个二维稀疏数值型数组。本类有 7 个属性和方法。

1. NumRows 属性

该属性保存数组的行维，NumRows 的值必须非负；如果值为 0，则行的编号取自 RowIndex 数组中的最大值。

2. NumColumns 属性

该属性保存数组的列维。

3. RowIndex 属性

该属性保存数组中非零元素的行编号数组。

4. ColumnIndex 属性

该属性保存数组中非零元素的列编号数组。

5. Array 属性

该属性保存稀疏数组中的非零值。

6. MWFlags 属性

该属性保存对象的一个引用。

7. CLone 方法

该方法创建对象的一个复制。

【例 9】 下面的 V B 代码创建一个 5×5 的对角稀疏矩阵。

```
X = [ 2   -1   0    0    0
     -1    2  -1    0    0
      0   -1   2   -1    0
      0    0  -1    2   -1
      0    0   0   -1    2 ]
Sub foo()
    Dim x As MWSparse
    Dim rows(1 To 13) As Long
    Dim cols(1 To 13) As Long
    Dim vals(1 To 13) As Double
    Dim I As Long, K As Long

    On Error GoTo Handle_Error
    K = 1
    For I = 1 To 4
        rows(K) = I
        cols(K) = I + 1
        vals(K) = -1
        K = K + 1
        rows(K) = I
        cols(K) = I
        vals(K) = 2
        K = K + 1
        rows(K) = I + 1
        cols(K) = I
        vals(K) = -1
        K = K + 1
    Next
    rows(K) = 5
    cols(K) = 5
    vals(K) = 2
    Set x = New MWSparse
    x.NumRows = 5
    x.NumColumns = 5
    x.RowIndex = rows
    x.ColumnIndex = cols
    x.Array = vals
    Exit Sub
Handle_Error:
    MsgBox (Err.Description)
End Sub
```

12.6.7 MWArg 类

该类传递一个 generic 变量到一个经过编译的方法。它包含 3 个属性和方法，即 Value As Variant, MWFlag As MWFlags 和 Clone 方法。

1. Value 属性

Value 属性保存要传递的当前变量。它可以是任意类型的。

2. MWFlags 属性

该属性保存一个 MWFlags 对象的引用。

3. CLone 方法

该方法创建一个对象的复制。

12.6.8 3 个枚举类型

工具类有 3 个枚举类型，即 mwArrayFormat, mwDataType 和 mwDateFormat。

（1）枚举 mwArrayFormat 为一系列数据转换时提醒数组格式化的常数。其常数值和描述如表 12-12 所示。

表 12-12 mwArrayFormat 枚举

常 数	值	描 述
mwArrayFormatAsIs	0	不进行格式化
mwArrayFormatMatrix	1	将数组格式化为矩阵
mwArrayFormatCell	2	将数组作为单元数组

（2）mwDataType 枚举用于提示变量的数据类型。该枚举的值和描述如表 12-13 所示。

表 12-13 mwDataType 枚举

常 数	值	MATLAB 类型
mwTypeDefault	0	N/A
mwTypeLogical	3	logical
mwTypeChar	4	char
mwTypeDouble	6	double
mwTypeSingle	7	single
mwTypeInt8	8	int8
mwTypeUint8	9	uint8
mwTypeInt16	10	int16
mwTypeUint16	11	uint16
mwTypeInt32	12	int32
mwTypeUint32	13	uint32

（3）mwDateFormat 枚举描述日期的不同格式。具体描述如表 12-14 所示。

表 12-14 mwDateFormat 枚举

常 数	值	描 述
mwDateFormatNameric	0	将日期定义为数值格式
mwDateFormatString	1	将日期定义为字符串格式

第 13 章　Excel 生成器（Excel Builder）

MATLAB 还提供了一个称为 Excel 生成器的工具，利用该工具，可以生成 DLL 组件和 VBA 代码。利用 DLL 组件，可以进行与前面 COM 生成器组件类似的操作。VBA 代码可以在 Excel 的 Visual Basic 编辑器中直接使用，可以保存为插件(Add-In)。

Excel 生成器创建的 COM 对象提供给 VB 程序环境一个类，该类包含一系列称为方法的函数，对应于包含在组件工程中的原始 MATLAB 函数。到目前为止，MATLAB 的 Excel 生成器组件只支持一个组件一个类。

13.1　创建 Excel 生成器插件

13.1.1　创建工程

要创建工程，在命令行中输入 MATLAB 命令 mxtool，显示 MATLAB 的 Excel 生成器主窗口，如图 13-1 所示。

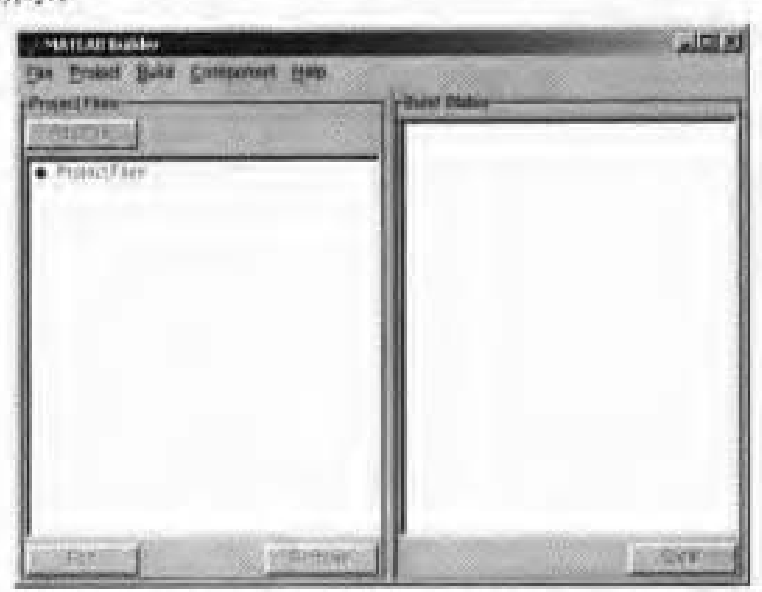


图 13-1　Excel 生成器主窗口

在窗口中依次选择 File→New Project，打开 "New Project Settings" 对话框，如图 13-2 所示。

1．"Component name" 文本框和 "Class name" 文本框

在 "Component name" 文本框中输入组件的名称，在 "Class name" 文本框中输入类的名称。组件名是后面创建的名称。输入组件名以后，生成器会自动输入一个与组件名相同的类名，可以将类名改为其他描述性更强的名称。

注意：*尽管组件名和类名可以相同，但组件名不能与后面添加的 M 文件或 MEX 文件的名字相同。*

2. “Project version” 文本框

在“Project version”文本框中输入组件的版本号。默认版本号为 1.0。

3. “Project directory” 文本框

在“Project directory”文本框中输入工程目录。工程目录指定编译和打包模型时，将工程和生成的文件放在那里。工程目录根据当前目录名和组件名自动创建。

注意：可以接受自动生成的工程目录路径或选择其他目录，一旦在菜单上单击了“OK”按钮就保存了目录。如果以后要移动工程或在它的路径上做任何改变，则需要重新进行整个工程的指定过程，包括添加文件到工程以及指定工程目录路径。

4. “Compiler options” 方框

如果选择“Create a singleton MCR”核选框，使用组件时只创建一个 MCR 实例。

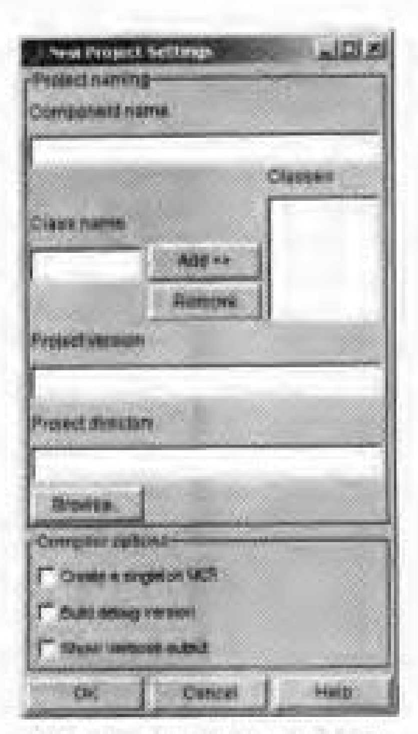


图 13-2 “New Project Settings”对话框

可以创建编译模型的一个调试版本，并能在调用 MATLAB 编译器时指定详细输出。该调试版本一方面允许跟踪，使出错报告显示 M 文件和发生错误的行。所有跟踪信息都会进行报告。如果不进行调试，则得不到 MATLAB 代码中发生错误的位置的指示。另一方面，它允许使用 Visual Studio 调试器进行完整的调试。

设置完以后，单击“OK”按钮，它们就成为工程工作空间的一部分并与添加到工程中的所有 M 文件和 MEX 文件一起被保存到工作空间中。一个名为<component_name>.cbl 的工程文件被自动添加到工程目录中。

13.1.2 管理 M 文件和 MEX 文件

创建工程以后，主窗口中的“Project”,“Build”和“Component”等菜单选项变为可用，如图 13-3 所示。单击“Add File”按钮或依次选择 Project→Add File，在工程中添加 M 文件和/或 MEX 文件。一次只能添加一个文件到工程中。

单击“Remove”按钮或依次选择菜单 Project→Remove File，将删除所有已经选定的 M 文件或 MEX 文件，一次可以删除多个文件。单击“Edit”按钮或依次选择 Project→Edit File…或双击 M 文件的文件名，将在 MATLAB 编辑器中打开选定的 M 文件，并可以进行修改和调试，但不能编辑 MEX 文件。

13.1.3 生成组件

定义工程设置和添加必要的 M 函数和 MEX 函数以后，可以生成一个可配置的 DLL 文件和必要的 VBA 代码。依次选择菜单选项 Build→Excel/COM Object，激活 MATLAB 编辑器，将中间的源文件写到<Project_dir>\src，将进行配置的输出文件写到<Project_dir>\distrib 目录。

“Build status”面板显示生成过程中的输出信息，报告遇到的任何问题。出现在

<Project_dir>\distrib 目录中的文件会是一个 DLL 文件和一个 VBA 文件(.bas)。生成的 DLL 会自动注册到系统中。

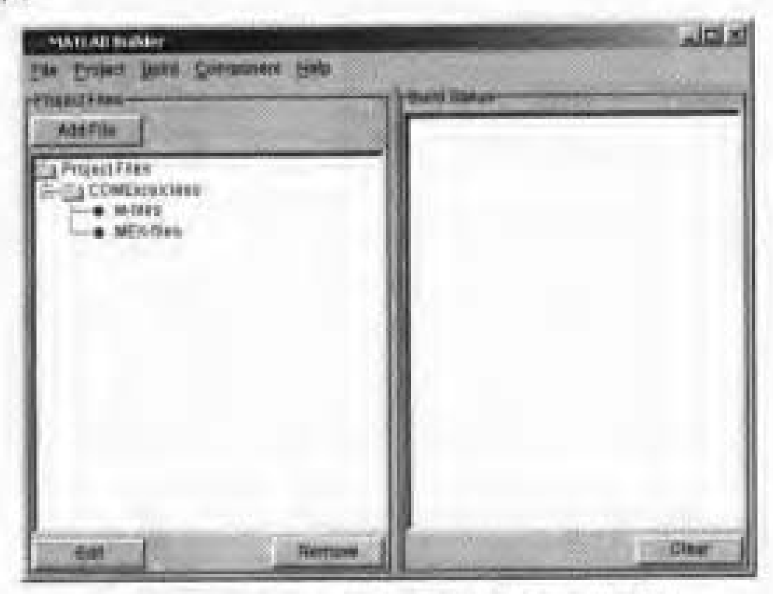


图 13-3　创建工程以后部分菜单和按钮变为可用

依次选择 Build→Clear Status，可以清除“Build Status”面板中的内容。生成过程的输出信息保存在文件<Project_dir>\build.log 中，要打开该 Log 文件，依次选择 Build→Open Build Log。Log 文件提供了一个生成过程的记录，在清除“Build status”面板中的内容以后还可以进行参考。

13.1.4　测试 VBA 模块

可以通过在 Excel 的 Visual Basic 编辑器中导入 VBA 文件和激活 Excel 工作表中的函数来测试模块。要把 VBA 代码导入到 Excel 的 Visual Basic 编辑器中，打开 Excel 并依次选择 File→Import，选择目录<Project_dir>\distrib 下已经创建的 VBA 文件。

创建工程时生成的 VB 模块包含必需的初始代码和 VBA 公式函数，每个公式函数包含了各自编译函数的调用，编译函数的格式可以通过 Excel 工作表中的单元得到。本函数考虑一个对于初始 MATLAB 函数输入的输入列表，返回一个对应于第 1 个输出参数的单一输出。这种类型的公式函数对于访问一个或多个输入的函数是最有用的。该函数返回一个标量值。

下面是根据前面生成的 VBA 代码创建插件的必要步骤。如果这些步骤不能工作，则引用创建.xls 文件的 Excel 文档。

（1）启动 Excel。

（2）依次选择菜单选项“工具”→“宏”→“Visual Basic 编辑器”。

（3）在“Microsoft Visual Basic”窗口中，依次选择“文件”→“导入文件…”。

（4）从<Project_dir>\dirstrib 目录选择 VBA 文件(.bas)。

（5）关闭 Visual Basic 编辑器。

（6）在 Excel 工作表窗口中，依次选择“文件”→“另存为…”。

（7）将另存类型设为 Microsoft Excel 加载宏。

（8）把.xla 文件保存到<Project_dir>\distrib 目录。

也可以将文件保存为*.xla 格式和*.bas 格式，要保存为*.xls 格式，按照上面的步骤进行。

但是将另存类型（第（5）步中）改为.xls 类型。保存为 VBA 代码，只进行（1）～（4）步。

13.1.5 打包和发布组件

如果模块编译成功，并且创建了 Excel 插件以后，则应准备包装组件，把它发布给终端用户。组件应该包括表 13-1 所示的文件。

表 13-1 组件应该包括的文件

文 件	说 明
_install.bat	用自解压可执行程序运行的脚本
<componentname_project ersion>.dll	经过编译的组件
MCRInstaller.exe	自解压的 MATLAB 组件运行时库工具，与平台有关
<componentname>.ctf	组件技术文档，与平台有关
*.xla	任何在目录中找到的插件文件

在目标机器上运行安装器按以下步骤进行：

（1）用 MCRInstaller 安装 MATLAB 组件运行时。

（2）要使用 Excel 插件，启动 Excel，选择“工具”→“加载宏...”，并选择需要的.xla 文件。

必须在每个要安装组件的计算机上重复本发布过程。

13.2 用 Excel 生成器组件编程

每个 MATLAB Excel 生成器组件都可作为一个独立的 COM 对象生成。通过 VBA 获取源于 Excel 的组件。本节提供了如何利用 VBA 编程环境将 MATLAB Excel 生成器组件插入 Excel 的一般信息。

可以用函数或过程创建一个简单的代码模块来很容易地将 MATLAB Excel 生成器组件插入 VBA 工程。这些函数或过程装载必要的组件，调用必要的方法，并且处理所有错误。

13.2.1 用 Excel 初始化生成器库

在使用任何 MATLAB Excel 组件前，用 Excel 的当前实例初始化支持的库。用工具库函数 MWInitApplication 来进行初始化，该函数是 MWUtil 类的一个成员，而 MWUtil 类是 MWComUtil 库的一部分。

添加初始化代码到 VBA 模块的一种途径是提供一个过程。该过程提供一次初始化，当发生调用时退出。下面的 VB 代码示例用当前 Excel 实例初始化库文件。一个名为 MCLUtil 的 Object 类型的全局变量控制 MWUtil 类的一个实例，另一个名为 bModuleInitialized 的 Boolean 型变量保存初始化过程的状态。

私有过程 InitModule 创建 MWComUtil 类的一个实例，并且用 Application 的一个变量调用 MWInitApplication 方法，一旦此函数完成，所有后面的调用将在不重新初始化的情况下退出。

本代码与创建组件时创建的 VBA 模块中的默认初始化代码相反，每个使用 MATLAB Excel 生成器组件的函数都可以在开始处包含一个对 InitModule 函数的调用，以保证初始化工作总是在必要的时候进行。

```
Dim MCLUtil As Object
Dim bModuleInitialized As Boolean

Private Sub InitModule()
      If Not bModuleInitialized Then
        On Error GoTo Handle_Error
        If MCLUtil Is Nothing Then
            Set MCLUtil = CreateObject("MWComUtil.MWUtil")
        End If
        Call MCLUtil.MWInitApplication(Application)
        bModuleInitialized = True
        Exit Sub
      Handle_Error:
        bModuleInitialized = False
      End If
End Sub
```

13.2.2 创建类的实例

调用类的方法以前，必须创建一个包含方法的类的实例，VBA 提供了两个技巧来做这件事情：

- CreateObject 函数；
- New 操作符。

1．CreateObject 函数

本方法使用 Visual Basic 应用程序接口（API）的 CreateObject 函数创建类的实例。要使用本方法，需要定义一个 Object 类型的变量作为类实例的引用，并将组件的 ProgID 作为变量调用 CreateObject 函数。如下例所示，假设有一个 ProgID 为“mycomponent.myclass.1_0”的组件，利用 CreateObject 函数创建该组件中 myclass 类的一个实例：

```
Function foo(x1 As Variant, x2 As Variant) As Variant
      Dim aClass As Object

    On Error Goto Handle_Error
    aClass = CreateObject("mycomponent.myclass.1_0")
    '(调用 aClass 的一些方法)
    Exit Function
Handle_Error:
    foo = Err.Description
End Function
```

2．New 操作符

本方法使用 New 操作符创建类的实例。在使用本方法以前，必须使用当前 VBA 工程中包含该类的类型库。要做到这一点，在 Visual Basic 编辑器中选择“工具”菜单，再选择“引用...”选项，显示“引用”对话框。然后，从该对话框中选择必要的类型库。

下面的例子演示如何使用 New 操作符创建一个类实例，假设在调用本函数以前从“可使用的引用”列表框中选择了“mycomponent 1.0 Type Library”选项。

```
Function foo(x1 As Variant, x2 As Variant) As Variant
    Dim aClass As mycomponent.myclass
    On Error Goto Handle_Error
    Set aClass = New mycomponent.myclass
    '(调用 aClass 的一些方法)
    Exit Function
Handle_Error:
    foo = Err.Description
End Function
```

本例中，类实例定义为 myclass，完整的表达形式为<component-name>.<class-name>，它保证在当前工程的其他库中包含名为 myclass 的类时不会出现名称冲突。

两种方法在函数上是等价的，第 1 种方法不需要引用 VBA 工程的类库，第 2 种方法可以获得更快的运行效果。

在前面两个例子中，用于方法调用的类型是一个过程的局部变量，它为每次调用创建和拆卸一个新的类实例，另外，也可以将它声明为一个模块级类实例，这样它就可以被所有的函数调用和重复使用，就像前面例子中的初始代码一样，如下所示。

```
Dim aClass As mycomponent.myclass

Function foo(x1 As Variant, x2 As Variant) As Variant
    On Error Goto Handle_Error
    If aClass Is Nothing Then
      Set aClass = New mycomponent.myclass
    End If
    '(调用 aClass 的一些方法)
    Exit Function
Handle_Error:
    foo = Err.Description
End Function
```

13.2.3 调用类实例的方法

创建类实例以后，可以调用类的方法来获取编译过的 MATLAB 函数。

当方法有输出参数时，第 1 个变量总是 nargout，它的数据类型为 Long，这个输入参数传递一般的 MATLAB nargout 参数给编译函数，并且指定要求有多少个输出参数。没有输出参数的方法不传递 nargout 参数。nargout 参数是输出参数，列在原始 MATLAB 函数的左侧，紧接着的输入参数列在右侧，所有输入和输出参数都是 Variant 类型的，为默认的 VB 数据类型。

一般地，除了用户自定义类型外，可以给类方法提供任何 VB 类型作为参数。也可以直接传递 Excel 的 Range 对象作为输入和输出参数。

下例演示从 VBA 向 MATLAB Excel 生成器组件类方法传递输入、输出参数的过程。

第 1 个例子是一个公式函数，它有两个输入，一个输出。

```
Function foo(x1 As Variant, x2 As Variant) As Variant
    Dim aClass As Object
    Dim y As Variant

    On Error Goto Handle_Error
    aClass = CreateObject("mycomponent.myclass.1_0")
```

```
    Call aClass.foo(1,y,x1,x2)
    foo = y
    Exit Function
Handle_Error:
    foo = Err.Description
End Function
```

第 2 个例子用过程代替原来的函数，并且用 Excel Range 作为输入和输出。

```
Sub foo(Rout As Range, Rin1 As Range, Rin2 As Range)
    Dim aClass As Object

    On Error Goto Handle_Error
    aClass = CreateObject("mycomponent.myclass.1_0")
    Call aClass.foo(1,Rout,Rin1,Rin2)
    Exit Sub
Handle_Error:
    MsgBox(Err.Description)
End Sub
```

13.2.4 处理 varargin 和 varargout 参数

当原始 MATLAB 函数中存在 varargin 和/或 varargout 时，这些参数将作为列表中最后的输入/输出参数被添加到类方法的参数列表中，可以创建一个 Variant 型数组。指定数组的元素为各个输入参数，并用该数组进行传递。

下面的例子创建一个 varargin 数组，以便调用一个由形如 y=foo(varargin)的 MATLAB 函数生成的方法。

```
Function foo(x1 As Variant, x2 As Variant, x3 As Varaint, _
            x4 As Variant, x5 As Variant) As Variant
    Dim aClass As Object
    Dim v(1 To 5) As Variant
    Dim y As Variant

    On Error Goto Handle_Error
    v(1) = x1
    v(2) = x2
    v(3) = x3
    v(4) = x4
    v(5) = x5
    aClass = CreateObject("mycomponent.myclass.1_0")
    Call aClass.foo(1,y,v)
    foo = y
    Exit Function
Handle_Error:
    foo = Err.Description
End Function
```

MWComUtil 工具库中的 MWUtil 类提供了一个 MWPack 帮助函数来创建 varargin 参数。

下面的例子将一个 varargout 参数分成 3 个 Excel Range 参数，foo 过程用到了工具库中的 MWUnpack 函数，用到的 MATLAB 函数为 varargout=foo(x1,x2)

```
Sub foo(Rout1 As Range, Rout2 As Range, Rout3 As Range, _
        Rin1 As Range, Rin2 As Range)
    Dim aClass As Object
    Dim aUtil As Object
    Dim v As Variant

    On Error Goto Handle_Error
    aUtil = CreateObject("MWComUtil.MWUtil")
    aClass = CreateObject("mycomponent.myclass.1_0")
    Call aClass.foo(3,v,Rin1,Rin2)
    Call aUtil.MWUnpack(v,0,True,Rout1,Rout2,Rout3)
    Exit Sub
Handle_Error:
    MsgBox(Err.Description)
End Sub
```

13.2.5 在调用方法的过程中控制错误

创建类实例或调用类方法时如果发生错误，将在当前过程中创建一个 Exception 标记。VB 在声明 On Error Goto <label>的过程中提供一个 Exception 块来控制兼容性，发生错误时跳至<label>行，所有的错误都通过这种方式来控制，包括原始 MATLAB 代码中的错误。

13.2.6 修改标记

每个 MATLAB Excel 生成器组件提供一个 MWFlags 类型的名为 MWFlags 的单一读写属性。MWFlags 属性由两部分组成：数组格式化标记和数据转换标记。数据转换标记改变数据从 Variant 型向 MATLAB 类型转换或相反时的行为。默认时，MATLAB Excel 生成器组件允许在类的水平上通过 MWFlags 类属性设置数据转换标记。本小节讨论如何设置这些标记。

1．数组格式化标记

数组格式化标记引导数据转换，以生成一个源于一般 Variant 型数据的 MATLAB 单元数组或矩阵作为输入，或生成一个 Variant 型数组或包含基本类型数组的单一 Variant 型数据作为输出。

下面的例子假设已经引用了当前工程中的 MWComUtil 库。

```
Sub foo( )
    Dim aClass As mycomponent.myclass
    Dim var1(1 To 2, 1 To 2), var2 As Variant
    Dim x(1 To 2, 1 To 2) As Double
    Dim y1,y2 As Variant

    On Error Goto Handle_Error
    var1(1,1) = 11#
    var1(1,2) = 12#
    var1(2,1) = 21#
    var1(2,2) = 22#
    x(1,1) = 11
    x(1,2) = 12
    x(2,1) = 21
```

```
        x(2,2) = 22
        var2 = x
        Set aClass = New mycomponent.myclass
        Call aClass.foo(1,y1,var1)
        Call aClass.foo(1,y2,var2)
        Exit Sub
    Handle_Error:
        MsgBox(Err.Description)
    End Sub
```

这里，两个 Variant 型参数，即 var1 和 var2 由相同的数组数据创建，但它们的结构不同。var1 是 2×2 的数组，每个元素包含一个 1×1 的 Double 型数据，而 var2 是一个 1×1 的 Variant 型参数，该参数包含一个 2×2 的值为 Double 型的数组。根据默认的数据转换原则，var1 转换为 2×2 的单元数组，每个单元为 1×1 的 Double 值，var2 直接转换为 2×2 的 Double 型矩阵。InputArrayFormat 标记控制转换这两个类型得到的公式。就像输出结果一样，前面例子中的两个数组都转换为 Double 型矩阵了，因为 InputArrayFormat 标记的默认值是 mwArrayFormatMatrix。使用本默认值是因为，就像输出结果一样，源于 Excel Range 的数组数据总是以 Variant 型数组的形式存在（就像前面例子中的 var1 一样），而且 MATLAB 函数经常处理矩阵参数。但是如果需要的是单元数组，则将 InputArrayFormat 标记设置为 mwArrayFormatCell。在创建类以后，调用方法以前添加下面的代码行：

```
aClass .MWFlags.ArrayFormatFlags.InputArrayFormat =
mwArrayFormatCell
```

类似地，可以用 OutputArrayFormat 标记改变输出参数的格式，也可以用 AutoResizeOutput 标记和 TransposeOutput 标记修改数组输出。

AutoResizeOutput 用于 Excel Range 对象，它被直接传递给输出参数。设置本标记以后，目标范围自动改变大小，以拟合生成的数组；如果没有设置，则目标范围必须至少与输出数组或数据大小一样。

TransposeOutput 标记转置所有的数组输出，当 MATLAB 函数的输出为一维数组时，这个标记很有用。默认时，MATLAB 把一维数组作为 $1\times n$ 的矩阵（行矢量），在 Excel 工作表中，它占据行的位置。

下面的例子自动改变矩阵大小并转置一个输出范围。

```
Sub foo(Rout As Range, Rin As Range )
    Dim aClass As mycomponent.myclass

    On Error Goto Handle_Error
    Set aClass = New mycomponent.myclass
    aClass.MWFlags.ArrayFormatFlags.AutoResizeOntput = True
    aClass.MWFlags.ArrayFormatFlags.TransposeOutput = True
    Call aClass.foo(1,Rout,Rin)
    Exit Sub
Handle_Error:
    MsgBox(Err.Description)
End Sub
```

2. 数据转换标记

数据转换标记处理数组元素的类型转换。CoerceNumericToType 和 InputDateFormat 两个数据转换标记控制数值和日期类型如何从 VBA 转换到 MATLAB。

考虑下面的例子：

```
Sub foo( )
    Dim aClass As mycomponent.myclass
    Dim var1, var2 As Variant
    Dim y As Variant

    On Error Goto Handle_Error
    var1 = 1
    var2 = 2#
    Set aClass = New mycomponent.myclass
    Call aClass.foo(1,y,var1,var2)
    Exit Sub
Handle_Error:
    MsgBox(Err.Description)
End Sub
```

本例将 Variant/Integer 型的 var1 转换为 int16 型，将 Variant/Double 型的 var2 转换为 Double 型，如果原始的 MATLAB 函数希望有两个 Double 型作为输入，上面的代码导致错误，解决的方法是将 var1 指定为 Double 型。但这可能很难办到。此时，将 CoerceNumericToType 标记设置为 mwTypeDouble，让数据转换器将所有的数值输入转换为 Double 型，在前面的例子中，创建类以后，调用方法前添加下面的代码行。

```
aClass .MWFlags.DataConversionFlags.CoerceNumericToType =
mwTypeDouble
```

InputDateFormat 标记控制 VBA Date 型数据的转换方式。本例将当前的日期和时间作为一个输入参数进行传递，并转换为字符串。

```
Sub foo( )
    Dim aClass As mycomponent.myclass
    Dim today As Date
    Dim y As Variant

    On Error Goto Handle_Error
    today = Now
    Set aClass = New mycomponent.myclass
    aClass. MWFlags.DataConversionFlags.InputDateFormat =
            mwDateFormatString
    Call aClass.foo(1,y,today)
    Exit Sub
Handle_Error:
    MsgBox(Err.Description)
End Sub
```

本例中第 1 个输出参数 y1 转换为 Date 型，第 2 个输出参数 y2 使用 aClass 提供的默认转换标记。

13.3 魔方示例

13.3.1 一个输入的情况

M 文件 mymagic 有一个整型输入，它指示所创建的魔方的大小。Excel 文件 mymagic.xls 用 3 种不同的方式使用它。

第 1 种方式调用函数 mymagic，参数为 4。该函数返回一个大小为 4 的魔方，并用该魔方填充一个 Excel 范围。

第 2 种方式使用转置标记对大小为 4 的魔方进行转置。

第 3 种方式更改输出的大小并在 Excel 工作簿中移动它。

注意：首先把<MATLAB>\toolbox\MATLABx1 \examples\xlmagic 中的目录 xlmagic 复制到<MATLAB>\work 中。

1．创建工程

从 MATLAB 命令行提示中将目录改变到<MATLAB>\work。输入命令 mxltool，启动 MATLAB Excel 生成器图形用户界面。从“File”菜单中选择“New Project”选项，打开“New Project Settings”对话框，如图 13-2 所示。

在“New Project Settings”对话框中按照下面的步骤进行设置。

（1）在“Component name”文本框中输入组件名“xlmagic”，通过单击“Tab”键将光标移到“Class name”文本框中。

（2）自动用名称“xlmagic”填充“Class name”文本框。

（3）版本号接受默认设置，即为 1.0。

（4）在“Project directory”文本框中输入组件的保存目录路径。默认时为<MATLAB>\work，后面添加组件名 xlmagic。其中，<MATLAB>为 MATLAB 的安装目录。可以改变目录，如果输入的目录不存在，则系统会提示你是否创建一个目录。

（5）单击“OK”按钮，创建 xlmagic 工程。

2．生成组件

生成组件按照以下步骤进行。

（1）从 Excel 生成器图形用户界面中单击“Add File...”按钮。

（2）从<MATLAB>\work\xlmagic 目录中选择文件 mymagic.m，并单击“Open”按钮。

（3）从“Build”菜单中单击“Build”或选择“Excel/COM Files”。

（4）给 Excel 中添加 Excel 生成器 COM 函数。

（5）启动 Excel。

（6）打开文件<MATLAB>\work\xlmagic\mymagic.xls。

3．组件应用演示

（1）演示在 Excel 中输出魔方

在 Excel 主窗口（不是 Visual Basic 编辑器）中同时选择“Alt”键和“F8”键，或从“工具”菜单中选择“宏”选项，然后从子菜单中选择“宏”选项。在列表中选择“mymagic”并单击“执行”按钮。本过程返回一个大小为 4 并从 B2 开始的魔方，如图 13-4 所示。

	A	B	C	D	E	F
1						
2	4	16	2	3	13	
3		5	11	10	8	
4		9	7	6	12	
5		4	14	15	1	
6						
7	The above example runs the macro "mymagic" which					
8	populates the cells B2 through E5 with a magic square of 4					
9	Select Tools->Macro-> Macros to run this example					

图 13-4　返回到 Excel 工作簿的魔方

（2）演示转置输出

重新打开“宏”对话框，选择 mymagic_transpose 宏并单击“执行”按钮。本过程中，一个大小为 4 的魔方发生转置，它的起始位置是 B14，如图 13-5 所示。

13					
14	4	16	5	9	4
15		2	11	7	14
16		3	10	6	15
17		13	8	12	1
18					
19	The above example runs the macro "mymagic_transpose" which				
20	transposes the results of a magic square of 4 and populates the				
21	cells B14 through E17				
22	Select Tools->Macro-> Macros to run this example				

图 13-5　转置后的魔方

（3）演示更改输出的大小

重新打开“宏”对话框，选择 mymagic_resize 宏，单击“执行”按钮。本过程返回一个大小为 4 的魔方，起始位置为 B32。将 A32 中的 4 改为一个更大的值，重新运行这个宏。返回一个刚刚指定大小的魔方，起始位置是 B32，如图 13-6 所示。

27	The below example runs the macro "mymagic_resize" which								
28	has an initial range for a magic square of 4 but will resize if								
29	the output is larger. Gradually increase the number in cell A32 and rerun the macro								
30	CAUTION: Resizing will over write any existing data in the target cells								
31									
32	8	64	2	3	61	60	6	7	57
33		9	55	54	12	13	51	50	16
34		17	47	46	20	21	43	42	24
35		40	26	27	37	36	30	31	33
36		32	34	35	29	28	38	39	25
37		41	23	22	44	45	19	18	48
38		49	15	14	52	53	11	10	56
39		8	58	59	5	4	62	63	1
40									

图 13-6　改变大小后的魔方

（4）察看 Visual Basic 代码

在 Excel 主窗口中，从“工具”菜单中选择“宏”菜单项，然后在子菜单中单击“Visual Basic 编辑器”选项，打开 Visual Basic 编辑器，如图 13-7 所示。

在 Visual Basic 编辑器中，在“工程-VBAProject”窗口中双击，展开工程 VBAProject(mymagic.xls)。展开 Modules 文件夹并双击 Module1 模块，打开“VB Code”窗口，显示本工程的代码。

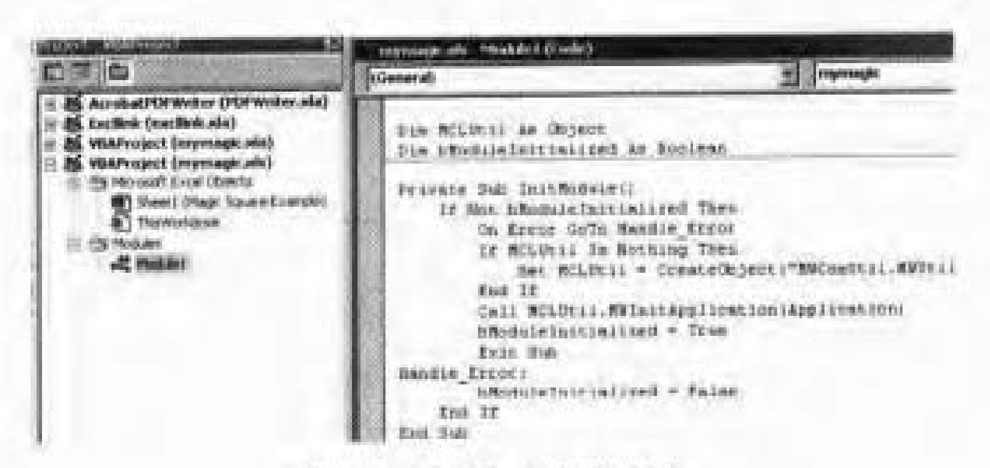


图 13-7　Visual Basic 编辑器窗口

13.3.2　使用多个文件和变量

（1）M 文件 myplot 有一个整型输入，功能是画一条从 1 到该数字的直线段。

（2）M 文件 mysum 有一个整型的 varargin 输入，功能是求所有数字的和并返回结果。

（3）M 文件 myprimes 有一个单精度整型输入 n，返回所有小于或等 n 的素数。

（4）Excel 文件 mymulti.xls 通过多种方式来演示这些 M 文件。

注意：开始前，将目录 xlmulti 从<MATLAB>\toolbox\MATLABxl\examples\xlmulti 复制到<MATLAB>\work。

1．创建工程

从 MATLAB 命令行提示中将目录改变为<MATLAB>\work，输入命令 mxltool，启动 MATLAB Excel 生成器图形用户界面。从"File"菜单中选择"New Project"，打开"New Project Settings"对话框。在该对话框中，按照下面的步骤进行设置。

（1）在"Component name"文本框中输入组件名 xlmulti。单击"Tab"键将光标移到"Class name"文本框中。

（2）此时"Class name"文本框中已经自动填充了类名 xlmulti，不改变它。

（3）版本按默认设置，为 1.0。

（4）在"Project directory"文本框中输入工程要保存的位置。

（5）单击"OK"按钮，创建 xlmulti 工程。

2．生成组件

按照以下步骤生成组件。

（1）在 Excel 生成器图形用户界面中单击"Add File..."按钮。

（2）从目录<MATLAB>\work\xlmulti 中选择文件 myplot.m，并单击"Open"按钮。

（3）重复上面的步骤，添加文件 myprimes.m 和 mysum.m。

（4）单击"Build"按钮或从"Build"菜单中选择"Excel/COM Files"选项。

（5）将 Excel 生成器 COM 函数添加到 Excel 中。

（6）启动 Excel。

（7）打开文件<MATLAB>\work\xlmulti\mymulti.xls，如图 13-8 所示。

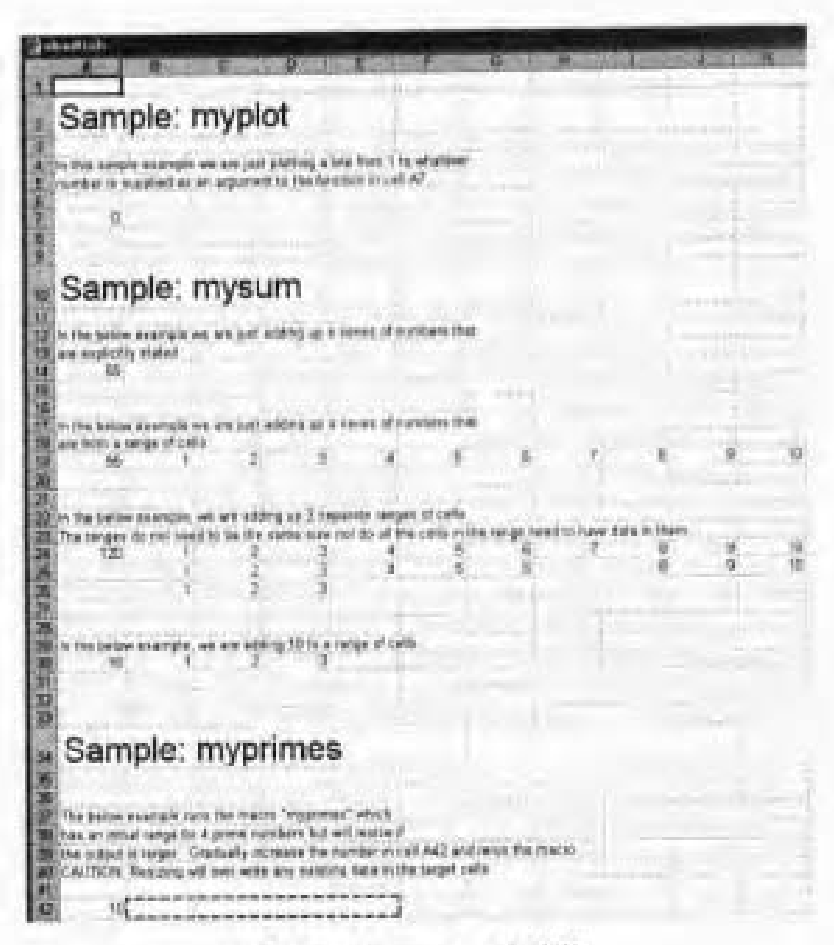


图 13-8 “mymulti.xls” 文件

3. 组件应用演示

（1）演示调用 myplot 函数

在本演示中，调用参数为 4 的 myplot 函数。要运行该函数，使 A7 成为激活的单元，单击“F2”，然后单击回车键，如图 13-9 所示。

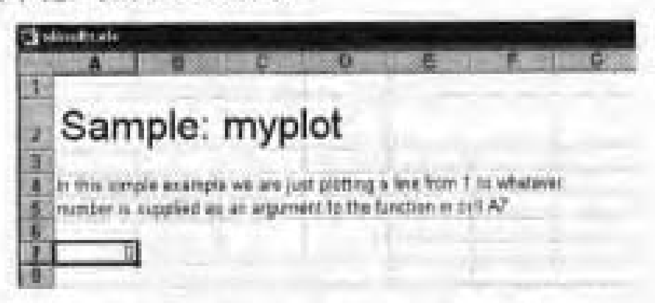


图 13-9 调用 myplot 函数（参数为 4）

本过程在 MATLAB 图形窗口中输入从 1 到 4 的直线段，如图 13-10 所示。

（2）演示用 4 种方式调用 mysum 函数

本演示通过 4 种方式调用 mysum 函数。第 1 种方式（单元 A14 中）求 1 到 10 的整数的和，返回结果 55。第 2 种方式（单元 A19）考虑单元范围，单元中的值从 1 到 10，返回结果 55。第 3 种方式（单元 A24）对几个范围对象相加，返回结果 120。第 4 种方式（单元 A30）考虑 range 对象和值，求它们的和并返回结果 16，如图 13-11 所示。

本函数在 Excel 文件打开时运行。

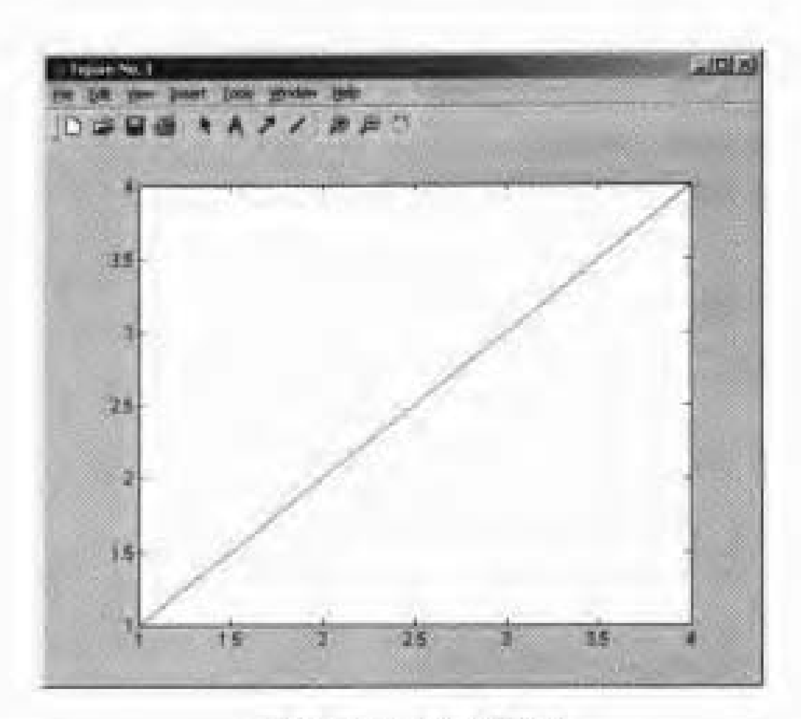


图 13-10 myplot 图形输出

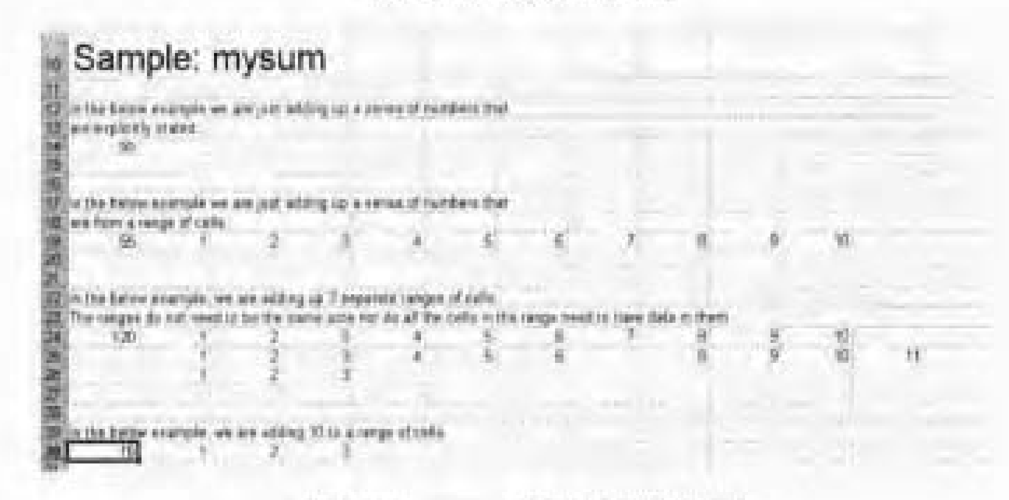


图 13-11 mysum 函数的 4 种调用方式

（3）演示 myprimes 宏

本演示中，宏 myprimes 调用函数 myprimes.m，初值为单元 A42 中的 10。该函数返回所有小于 10 的素数到 B42～E42 的单元中，如图 13-12 所示。

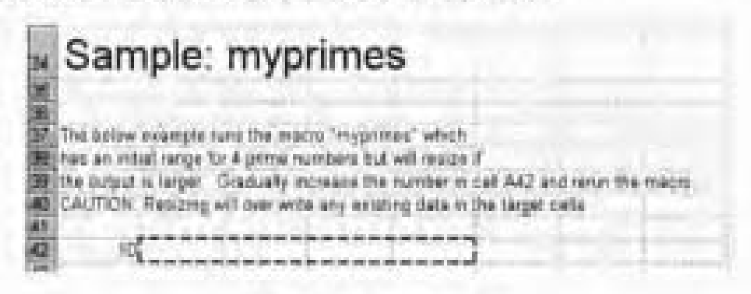


图 13-12 myprimes 宏

要运行本宏，在 Excel 主窗口（而不是 Visual Basic 编辑器）中同时按下"Alt"键和"F8"键，或从"工具"菜单中选择"宏"选项，然后从子菜单中选择"宏"选项。从列表中选择

"myprimes" 并单击 "执行" 按钮。结果如图 13-13 所示。

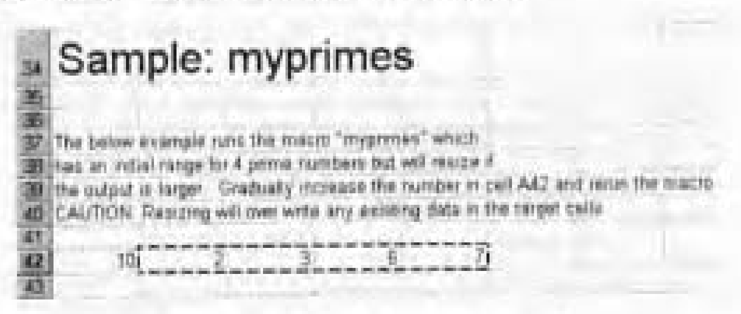


图 13-13 值等于 10 时 myprimes 的输出

如果输出内容大于指定的输出范围，则本函数会自动改变大小，将单元 A42 中的值改变为一个大于 10 的值，然后重新运行宏，把所有小于该值的素数返回到单元 A42 中，如图 13-14 所示。

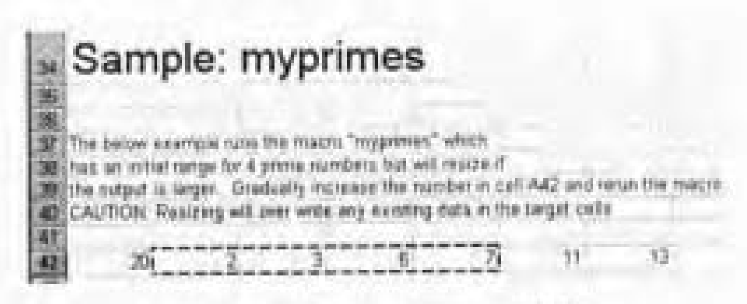


图 13-14 值大于 10 时 myprimes 的输出

（4）察看 Visual Basic 代码

察看 Visual Basic 代码按照以下步骤进行。

① 在 Excel 中，从 "工具" 菜单中选择 "宏" 菜单，然后在其子菜单中选择 "Visual Basic 编辑器" 选项。

② 在 Visual Basic 编辑器中的 "工程-VBAProject" 窗口内展开工程 "VBAProject (mymulti.xls)"。

③ 展开 "Mocules" 文件夹并双击 "Module1" 模块，打开 VB 代码窗口。窗口中显示本工程的所有代码，如图 13-15 所示。

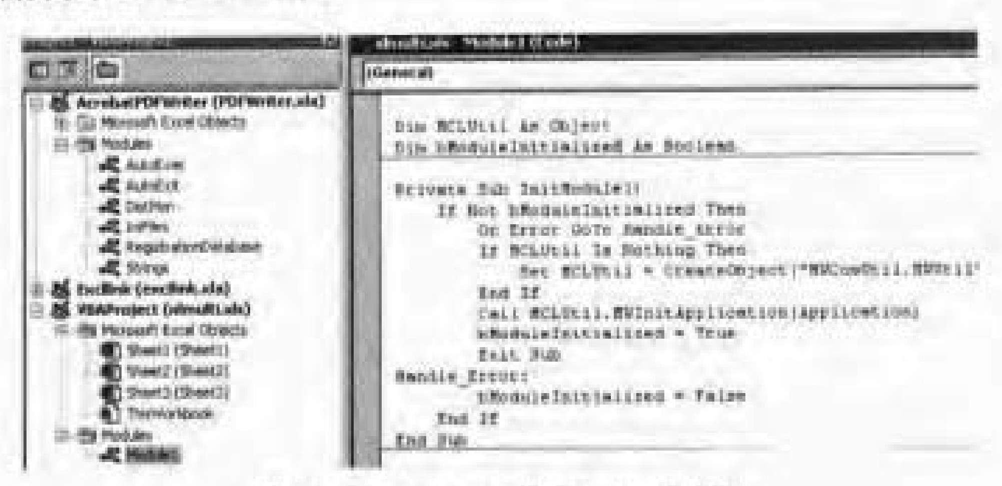


图 13-15 mymulti.xls 的 Visual Basic 代码

13.4 谱分析示例

本示例演示如何通过创建一个复杂的Excel插件来进行谱分析。演示这个实例，需要Visual Basic窗体和控件以及Excel工作簿事件方面的知识。

本例通过创建一个Excel插件来演示快速傅立叶变换（FFT），其输入数据从指定的Excel工作簿范围中获取。本函数返回一个FFT结果、一个频率点数组和输入数据的功率谱密度。它把结果放到当前工作簿中指定的范围内。可以有选择地用图形显示功率谱密度。可以从Excel的“工具”菜单中调用开发的函数，并通过图形用户界面来指定输入和输出。

创建Excel插件按照以下步骤进行。

（1）从MATLAB代码中创建一个独立的COM组件。

（2）实现必要的VBA代码来完成输入和对组件的调用。

（3）创建图形用户界面。

（4）创建一个Excel插件并对所有必要的组件进行打包。

13.4.1 创建组件

要创建的组件有两种方法：computefft和plotfft。computefft方法计算FFT和输入数据的功率谱密度，并计算基于所输入数据的长度和采样间隔的频率点矢量。plotfft方法完成同样的操作，另外还在MATLAB图形窗口中用图形显示输入的数据和功率谱密度。这两种方法的代码分别放在两个M文件中，即computefft.m和plotfft.m。

computefft.m:

```
function [fftdata, freq, powerspect] = computefft(data, interval)
    if (isempty(data))
        fftdata = [];
        freq = [];
        powerspect = [];
        return;
    end
    if (interval <= 0)
        error('Sampling interval must be greater then zero');
        return;
    end
    fftdata = fft(data);
    freq = (0:length(fftdata)-1)/(length(fftdata)*interval);
    powerspect = abs(fftdata)/(sqrt(length(fftdata)));
```

plotfft.m:

```
function [fftdata, freq, powerspect] = plotfft(data, interval)
    [fftdata, freq, powerspect] = computefft(data, interval);
    len = length(fftdata);
    if (len <= 0)
        return;
    end
    t = 0:interval:(len-1)*interval;
    subplot(2,1,1), plot(t, data)
```

```
xlabel('Time'), grid on
title('Time domain signal')
subplot(2,1,2), plot(freq(1:len/2), powerspect(1:len/2))
xlabel('Frequency (Hz)'), grid on
title('Power spectral density')
```

按照以下步骤创建组件。

（1）启动 comtool。

（2）用下面的设置创建一个新的工程：

组件名：Fourier

类名：Fourier

工程版本：1.0

选择“Use Handle Graphics library”核选框

（3）把 computefft.m 和 plotfft.m 添加到工程中。

（4）保存工程。

（5）单击“Build”按钮，完成创建组件。

13.4.2 将组件集成到 VBA 中

组件建好以后，可以实现必要的 VBA 代码，将它集成到 Excel 中去。按照下面的步骤打开 Excel 并选择必要的库文件开发插件。

（1）启动 Excel。

（2）从 Excel 主菜单中，按顺序选择“工具”→“宏”→“Visual Basic 编辑器”，启动 Excel 的 Visual Basic 编辑器。

（3）在编辑器的“工具”菜单中选择“引用...”选项，显示“引用”对话框。在该对话框中的列表框中选择“Fourier 1.0 Type Library”和“MWComUtil 1.0 Type Library”。

插件需要一些初始化代码和一些全局变量来控制应用程序的状态。要达到这个目的，按照下面的步骤实现一段 Visual Basic 代码模块。

（4）在工程窗口中右击 VBAProjec 选项并从弹出式菜单中按顺序选择“插入”→“模块”。

（5）在 VBAProject 中的 Modules 下面出现一个新的模块。在该模块的属性页中，将 Name 属性设置为 FourierMain，如图 13-16 所示。

在 FourierMain 模块中输入下面的代码：

```
' FourierMain 模块一为主模块，它提供全局变量的声明并提供初始化代码
'
Public theFourier As Fourier.Fourier       'Fourier 对象的全局实例
Public theFFTData As MWComplex             'MWComplex 对象的全局实例
Public InputData As Range           '输入数据的范围
Public Interval As Double           '采样间隔
Public Frequency As Range           '输出频率数据范围
Public PowerSpect As Range          '输出功率谱密度范围
Public bPlot As Boolean             '控制绘图状态
Public bInitialized As Boolean      '控制模块是否被初始化

Private Sub LoadFourier()
    '初始化全局变量并载入 Spectral Analysis 窗体
    Dim MainForm As frmFourier
```

```
    On Error GoTo Handle_Error
    Call InitApp
    Set MainForm = New frmFourier
    Call MainForm.Show
    Exit Sub
Handle_Error:
    MsgBox (Err.Description)
End Sub

Private Sub InitApp()
'初始化类和库
'运行一次当前 Excel 进程
If bInitialized Then Exit Sub
  On Error GoTo Handle_Error
  If theFourier Is Nothing Then
    Set theFourier = New Fourier.Fourier
  End If
  If theFFTData Is Nothing Then
    Set theFFTData = New MWComplex
  End If
  bInitialized = True
  Exit Sub
Handle_Error:
  MsgBox (Err.Description)
End Sub
```



图 13-16　在 VBA 工程中插入模块

13.4.3　创建图形用户界面

集成处理的下一步是用 Visual Basic 编辑器为插件开发一个用户界面。按照以下步骤创建一个新的用户窗体，并在窗体上添加必要的控件。

（1）在工程窗口中右击 VBAProject 选项，并从弹出式菜单中按顺序选择“插入”→“用户窗体”选项。

（2）现在，在 VBA 工程中，一个新的窗体显示在 Forms 下面。在窗体的属性页中，将 Name 属性设置为 frmFourier，将 Caption 属性设置为 Spectral Analysis。

（3）给空白的窗体中添加控件，如图 13-17 所示，并按表 13-2 中的内容设置控件的属性。



图 13-17　创建 Visual Basic 窗体

表 13-2　属性设置

控件类型	控件名称	属　性
Frame	Frame1	Caption =Input Data
Label	Label1	Caption = Input Data:
RefEdit	refedtInput	
Label	Label2	Caption =Sampling Interval
CheckBox	chkPlot	Caption =Plot time domain Signal and Power Spectral Density
Frame	Frame2	Caption =Output Data
Label	Label3	Caption =Frequency:
RefEdit	refedtFreq	
Label	Label4	Caption = FFT-Real Part:
RefEdit	refedtReal	
Label	Label5	Caption = FFT-Imaginary Part:
RefEdit	refedtImag	
Label	Label6	Caption =Power Spectral Density
RefEdit	refedtPowSpect	
CommandButton	btnOK	Caption = OK Default = True
CommandButton	btnCancel	Caption = Cancel Cancel = True

窗体和控件创建完毕以后，右击窗体并从弹出式菜单中选择“查看代码”选项。然后在代码窗口中输入下面的代码。

```
'frmFourier 事件句柄
'
Private Sub UserForm_Activate()
'UserForm 被激活时的事件句柄。本函数在显示窗体以前被调用，
'并用保存在全局变量中的值初始化所有控制
    On Error GoTo Handle_Error
    If theFourier Is Nothing Or theFFTData Is Nothing Then Exit Sub
    '用当前状态初始化控制
    If Not InputData Is Nothing Then
        refedtInput.Text = InputData.Address
    End If
    edtSample.Text = Format(Interval)
    If Not Frequency Is Nothing Then
        refedtFreq.Text = Frequency.Address
    End If
  If Not IsEmpty (theFFTData.Real) Then
  If IsObject(theFFTData.Real) And TypeOf theFFTData.Real Is Range Then
      refedtReal.Text = theFFTData.Real.Address
        End If
    End If
  If Not IsEmpty (theFFTData.Imag) Then
  If IsObject(theFFTData.Imag) And TypeOf theFFTData.Imag Is Range Then
      refedtImag.Text = theFFTData.Imag.Address
        End If
    End If
    If Not PowerSpect Is Nothing Then
        refedtPowSpect.Text = PowerSpect.Address
    End If
    chkPlot.Value = bPlot
    Exit Sub
Handle_Error:
    MsgBox (Err.Description)
End Sub

Private Sub btnCancel_Click()
'单击“Cancel”按钮时的事件句柄。在不计算 fft 或更新变量的情况下退出窗体
    Unload Me
End Sub
Private Sub btnOK_Click()
'单击“OK”按钮时的事件句柄。在控件中更新所有变量的值，
'并运行 computefft 或 plotfft 方法
    Dim R As Range

    If theFourier Is Nothing Or theFFTData Is Nothing Then GoTo Exit_Form
    On Error Resume Next
    '输入
    Set R = Range(refedtInput.Text)
```

```
        If Err <> 0 Then
            MsgBox ("Invalid range entered for Input Data")
            Exit Sub
        End If
        Set InputData = R
        Interval = CDbl(edtSample.Text)
        If Err <> 0 Or Interval <= 0 Then
            MsgBox ("Sampling interval must be greater than zero")
            Exit Sub
        End If
        'Process Outputs
        Set R = Range(refedtFreq.Text)
        If Err = 0 Then
            Set Frequency = R
        End If
        Set R = Range(refedtReal.Text)
        If Err = 0 Then
            theFFTData.Real = R
        End If
        Set R = Range(refedtImag.Text)
        If Err = 0 Then
            theFFTData.Imag = R
        End If
        Set R = Range(refedtPowSpect.Text)
        If Err = 0 Then
            Set PowerSpect = R
        End If
        bPlot = chkPlot.Value
        '计算 fft 并有选择地绘制功率谱密度图
        If bPlot Then
            Call theFourier.plotfft(3, theFFTData, Frequency, PowerSpect,_
              InputData, Interval)
        Else
            Call theFourier.computefft(3, theFFTData, Frequency, PowerSpect,_
              InputData, Interval)
        End If
        GoTo Exit_Form
    Handle_Error:
        MsgBox (Err.Description)
    Exit_Form:
        Unloed Me
    End Sub
```

集成的最后一步是给 Excel 添加一个菜单选项。这样，就可以从 Excel 的“工具”菜单中调用该工具了。进行这一步工作，需要为工作簿的 AddinInstall 和 AddinUninstall 事件添加事件句柄。该菜单项调用 LoadFourier 模块中的 LoadFourier 函数。按照下面的步骤实现菜单选项。

（1）在 Visual Basic 工程窗口中右击“ThisWorkbook”选项，并从弹出式菜单中选择“查看代码”选项，结果如图 13-18 所示。

（2）将下面的代码放到 ThisWorkbook 对象中。



图 13-18 向 Excel 中添加一个菜单项

```
Private Sub Workbook_AddinInstall()
'安装插件时被调用
    Call AddFourierMenuItem
End Sub

Private Sub Workbook_AddinUninstall()
'卸载插件时被调用
    Call RemoveFourierMenuItem
End Sub

Private Sub AddFourierMenuItem()
    Dim ToolsMenu As CommandBarPopup
    Dim NewMenuItem As CommandBarButton

    '如果已经存在，则删除它
    Call RemoveFourierMenuItem
    '找到"Tools"菜单
    Set ToolsMenu = Application.CommandBars(1).FindControl(ID:=30007)
    If ToolsMenu Is Nothing Then Exit Sub
    '添加谱密度分析菜单选项
    Set NewMenuItem = ToolsMenu.Controls.Add(Type:=msoControlButton)
    NewMenuItem.Caption = "Spectral Analysis..."
    NewMenuItem.OnAction = "LoadFourier"
End Sub

Private Sub RemoveFourierMenuItem()
  Dim CmdBar As CommandBar
  Dim Ctrl As CommandBarControl
  On Error Resume Next
  '找到"Tools"菜单并删除 Spectral Analysis 菜单选项
  Set CmdBar = Application.CommandBars(1)
  Set Ctrl = CmdBar.FindControl(ID:=30007)
  Call Ctrl.Controls("Spectral Analysis...").Delete
End Sub
```

其中，Workbook_AddinInstall 过程调用 AddFourierMenuItem 过程，在窗体中添加菜单项，它在安装插件时被调用；Workbook_AddinUninstall 过程在卸载插件时被调用；AddFounierMenuItem 过程在“Tools”菜单中添加一个子菜单项；RemoveFourierMenuItem 过程把 Spectral Analysis 菜单项从“Tools”菜单中删除。

13.4.4 保存和测试插件

（1）将插件命名为“Spectral Analysis”，按照下面的步骤保存它：

- 在 Excel 主菜单中依次选择“文件”→“属性”；
- 当“Workbook 属性”对话框出现时，选择“摘要信息”选项卡，并输入“Spectral Analysis”作为工作簿的标题；
- 单击“OK”，保存编辑内容；
- 从 Excel 主菜单中依次选择“文件”→“另存为...”；
- 在“另存为”对话框中选择“Microsoft Excel Add-In(*.xla)”作为文件类型；
- 输入“Fourier.xla”作为文件名，并单击“保存”按钮，保存插件。

在分发插件以前，对它进行测试。谱分析常用于查找淹没于噪声时间段信号中的频率组分。本例中，将创建一套表示有两个完全不同组分的信号的数据，并给它添加一个随机成分。本数据与输出一起，将被保存在 Excel 工作簿的列中，将时间域信号与功率谱密度一起用图形表示。

（2）按照以下步骤创建测试程序：

- 启动一个新的 Excel 进程，它有一个空白的工作簿；
- 从主菜单中依次选择“工具”→“加载宏...”，打开“加载宏”对话框；
- 在“加载宏”对话框中单击“浏览...”按钮；
- 找到 Fourier.xla 文件并单击“确定”按钮；
- 在列表框中找到“Spectral Analysis”插件，选择它；
- 单击“OK”按钮，装载插件。

本插件在 Excel 的“工具”菜单下安装一个菜单项。可以通过依次选择“工具”→“Spectral Analysis”来显示谱分析的图形用户界面。调用插件以前，创建一些数据，本例中，信号的组分为 15Hz 和 40Hz。对信号进行 10 秒钟的采样，每隔 0.01 秒采样一次。将时间点放在 A 列，信号点放在 B 列。

（3）按照下面的步骤创建数据：

- 在当前工作表中，在单元 A1 中输入 0；
- 单击单元 A2 并键入公式“=A1+0.01”；
- 单击并一直按住单元 A2 的右下角并把公式向下拖至 A1001。本过程用以 0.01 为增量 0 到 1 之间的数充填 A1：A1001 范围；
- 单击单元 B1，并键入公式“=SIN(2*PI()*15*A1)+SIN(2*PI()*40*A1)+RAND()”，重复拖拉过程，将该公式复制到 B1：B1001 范围内的所有单元。

（4）使用数据的列（B 列）对插件作如下测试：

- 从主菜单中依次选择“工具”→“Spectral Analysis...”；
- 单击“Input Data”窗口；
- 在工作簿中选择 B1：B1001 范围或将本地址键入到“Input Data”窗口中；
- 在“Sampline Interval”窗口上单击并键入 0.01；

• 选择"Plot time domain signal and power spectral density"；

• 输入 C1:C1001，用于频率输出，并类似地输入 D1:D1001，E1:E1001 和 F1:F1001，分别用于 FFT 的实部和虚部以及谱密度；

• 单击"OK"按钮进行分析。

图 13-19 显示了测试的结果。

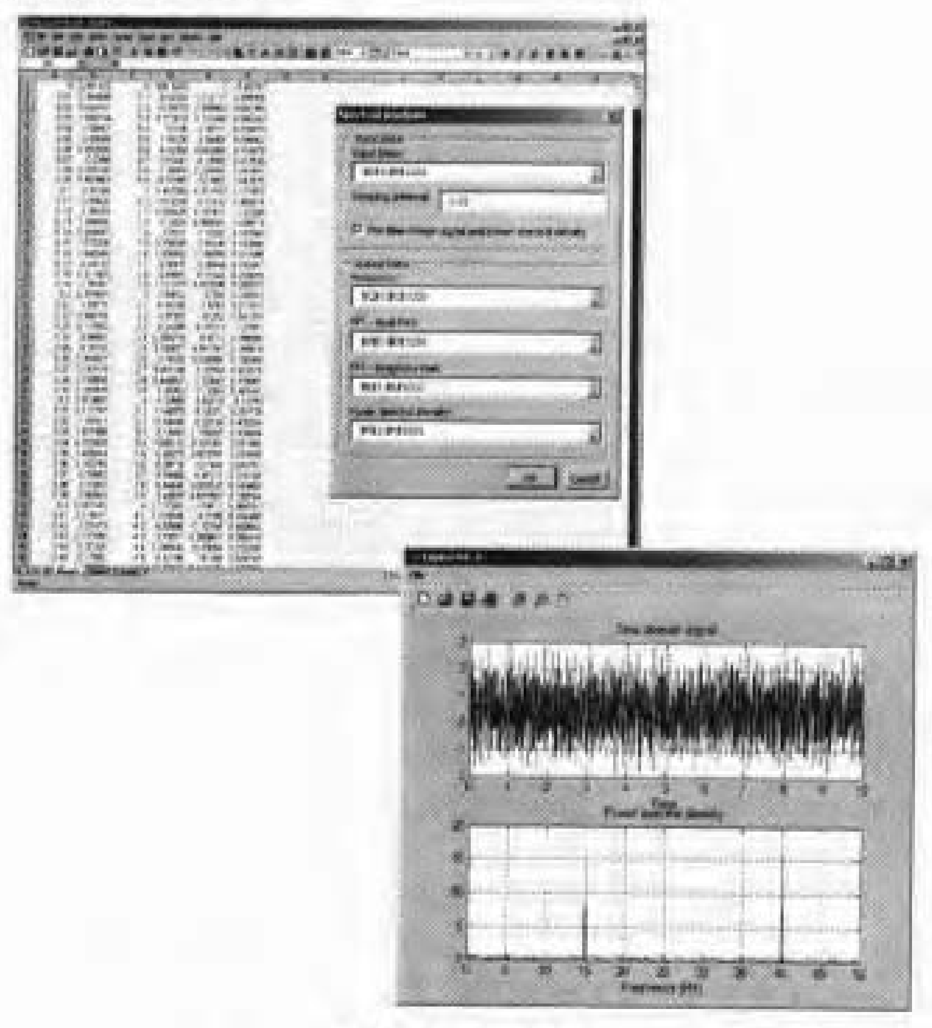


图 13-19 测试的结果

功率谱密度展示了 15Hz 和 40Hz 处的两个信号。

13.4.5 打包组件

将 COM 组件和所有的支持库打包到一个自解压可执行程序中。这样，本包就可以安装到其他需要使用 Spectral Analysis 组件的计算机上了。同样需要将 Fourier.xla 文件复制到所有在 Excel 中使用本组件的计算机中。打包组件按照以下步骤进行。

（1）回到 COM 生成器卡界面。如果已经关闭，重新启动它并装载 Fourier 工程。

（2）依次选择 Component→Package Component。

本命令创建 Fourier.exe 自解压可执行程序。要把本组件安装到另一台计算机上，将 Fourier.exe 包复制到那台计算机上，从命令行中运行它。

第 14 章　MATLAB 与硬件接口

MATLAB 提供了与其他硬件之间的接口函数及方法，就如同在 VC 及 VB 中的基本类库一样；使得外设和 MATLAB 之间可以直接方便地进行通信，基本的通信方式有并行通信接口和串行通信接口两种。本章将对 MATLAB 串行通信接口进行详细的介绍。

14.1　MATLAB 串行通信接口简介

本节将简单介绍 MATLAB 串行通信接口的概念及所支持的串行通信接口标准。

14.1.1　什么是 MATLAB 串行通信接口

使用 MATLAB 串行通信接口（简称串口），可以直接访问外设，如 modem 和与计算机串口相连的科学仪器。MATLAB 串口通过串口对象来建立。用串口对象支持函数和串口对象属性函数，能实现下面的功能：

- 配置串口通信；
- 读写数据；
- 用事件和回调函数；
- 记录信息到磁盘。

如果想和获取数据的硬件（如多功能 I/O 板）通信，需要数据获取工具箱；如果想和 GPIB（通用接口总线）或者 VISA 兼容器件通信，需要器件控制工具箱。当然，这些工具箱也包括其他串口 I/O 实用函数，使得可以很方便地产生和配置对象以及进行通信等。

14.1.2　支持的串行通信接口标准及平台

现行的串行通信标准包括 RS-232, RS-422 和 RS-485, MATLAB 串口对象支持所有这些标准。在这些标准中，通常使用 RS-232 来连接计算机和外设。下面的分析将以 RS-232 为标准进行。

MATLAB 串口所支持的操作系统为 Microsoft Windows，Linux 及 Sun Solaris。

14.2　进一步了解串行接口

本节将对串行接口进行详细的介绍，包括接口标准、信号及管脚分配、如何用电缆连接通信设备及查找系统串行接口信息并正确配置的方法。

14.2.1　什么是串行通信

与外界的信息交换称为通信。基本的通信方式有并行通信和串行通信两种。

一条信息的各位数据被同时传送的通信方式称为并行通信。并行通信的特点是：各数据位同时传送，传送速度快、效率高，但有多少数据位就需多少根数据线，因此传送成本高，

所以只适用于近距离（相距数米）的通信。

一条信息的各位数据被逐位按顺序传送的通信方式称为串行通信。串行通信的特点是：以串行的方式（一次一个比特）发送和接收字节信息，字节以二进制或文本的格式被传送，最少只需一根传输线即可完成，成本低但传送速度慢。串行通信的距离可以从几米到几千米。

根据信息的传送方向，串行通信可以进一步分为单工、半双工和全双工三种。信息只能单向传送为单工；信息能双向传送但不能同时双向传送称为半双工；信息能够同时双向传送则称为全双工。

进行串行通信的设备通常一个是计算机，而另一个是modem、另一台计算机或科学仪器，如示波器或函数发生器等。

14.2.2 串行接口标准

连接两个设备的串行接口由美国电信工业协会颁布的 TIA/EIA-232C 标准指定。最初的串行接口标准为 RS-232，现在仍广泛在使用。RS-232 定义了几个串口特征：

- 最大比特传输率及电缆长度；
- 信号名、电气特性及功能；
- 连接及管脚分配；

基本的通信只需要收、发和地三条管脚即可以完成，其它管脚可选作为数据流控制，不是必需的。

14.2.3 串行接口信号及管脚分配

串口包含两类信号：数据信号及控制信号。为了支持这些信号类型及信号地，RS-232 标准定义了一种 25 针的管脚连接，但是，大多数的 PC 机及 UNIX 平台用 9 针连接。实际上，完成通信只需要上面说的三根针脚：一个收、一个发及一个信号地。

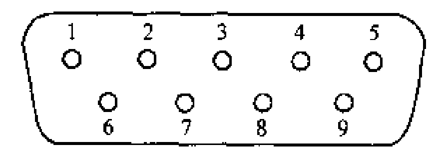


图 14-1　9 针公头针脚分配框图

数据终端设备DTE（计算机）的9针公头针脚分配框图如图14-1所示。

接口引线及功能如表 14-1 所示。

表 14-1　接口引线及功能

引 脚 号	名称（缩写名）	功 能 说 明
1	载波检测（CD）	当 DCE 从数据线上收到信号时，发出此信号
2	接收数据（RXD）	从 DCE（数据通信设备）到 DTE 的数据线
3	发送数据（TXD）	从 DTE 到 DCE 的数据线
4	数据终端就绪（DTR）	由 DTE 发出，表示 DTE 可以传数
5	信号地（GND）	用于接口的逻辑地
6	外设就绪（DSR）	由 DCE 发出，表示数据已被接收
7	请求发送（RTS）	由 DTE 发出，DCE 根据情况决定响应否
8	允许发送（CTS）	由 DCE 发出，控制发送端发送数据
9	振铃指示	由 DCE 发出，表示正进行通信

14.2.4 用串行电缆连接通信设备

RS-232 标准定义了用电缆连接两个设备，即数据终端设备 DTE（如计算机）及数据通信

设备 DCE（如 modem）。因为数据通信设备（DCE）为 2 脚发，3 脚收。所以直接电缆连接 DTE 和 DCE 的连接图如图 14-2 所示：

如果用电缆连接两个 DTE 或者 DCE，需要交叉连接线。用电缆连接两个 DTE 如图 14-3 所示。此处交叉是因为 DTE 都是 2 脚收，3 脚发。

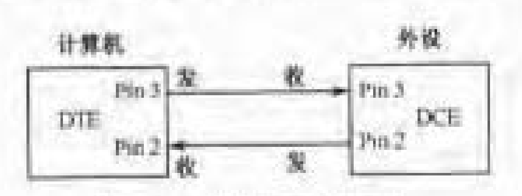


图 14-2　连接 DTE 和 DCE

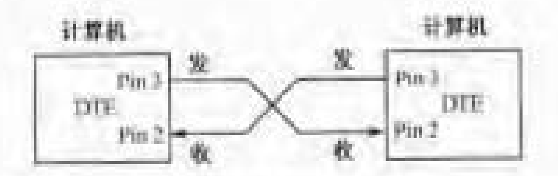


图 14-3　连接 DTE 和 DTE

14.2.5　查找所使用平台的串行接口信息

下面将介绍查找 Windows 和 UNIX 系统中串口信息的方法。系统为所有串口设置了默认值。但是通过 MATLAB 的代码重载，默认值对于串口应用程序将无效，必须单独加以设置。

1．Windows 平台

可以很容易地通过 Windows 控制面板来获得串口信息，按照下面的步骤打开控制面板：开始→设置→控制面板。

对于 Windows NT，选择串口图标并打开来获得串口清单，对话框如图 14-4 所示。

为了获得 COM1 的设置信息，选择列表框中的“COM1”选项，并且单击“Settings...”按钮，打开“Settings for COM1”对话框，如图 14-5 所示。



图 14-4　“Ports”对话框



图 14-5　“Settings for COM1”对话框

2．UNIX 平台

要发现 UNIX 平台的串口信息，需要知道串口名，这些名字可能因为操作系统不同而不同。

（1）在 Linux 系统中，为 ttyS0, ttyS1。可以通过 setserial 命令来显示或者配置串口信息。例如，显示哪些串口可用：

```
setserial -bg /dev/ttyS*
```

返回为：

```
/dev/ttyS0 at 0x03f8 (irq = 4) is a 16550A
/dev/ttyS1 at 0x02f8 (irq = 3) is a 16550A
```

显示 ttyS0 的详细信息：

```
setserial -ag /dev/ttyS0
```

返回为：

```
/dev/ttyS0, Line 0, UART: 16550A, Port: 0x03f8, IRQ: 4
Baud_base: 115200, close_delay: 50, divisor: 0
closing_wait: 3000, closing_wait2: infinte
Flags: spd_normal skip_test session_lockout
```

如果 setserial -ag 命令不起作用，请确认是否有串口读写权限。

（2）对于所有支持的 UNIX 平台，可以使用 stty 命令来显示或者配置串口信息。

例如，显示串口 ttyS0 信息如下：

```
stty -a < /dev/ttyS0
```

配置波特率为 4800b/s

```
stty speed 4800 < /dev/ttyS0 > /dev/ttyS0
```

14.3 用串行接口进行通信

为了方便调试，在下面的例子中，我们假定外设是另一台计算机，我们便可以在一台计算机上调试通信行为。使用标准接口线，将其中一端的 2, 3 脚交换，将其连接于计算机的两个串口，一端发，另一端收。我们利用程序"串口调试助手"作为接收显示程序（当然也可以利用 Windows 的超级终端），对于其他的外设，通信是类似的。

14.3.1 一个简单的例子

如果我们假定串口为 COM1，COM2 为外设的收端，传输率为 4800b/s，用下面的程序可以完成通信。

```
s = serial('COM1');
set(s,'BaudRate',4800);
fopen(s);
fprintf(s,'*IDN?')
out = fscanf(s);
fclose(s)
delete(s)
clear s
```

用"串口调试助手"通过 COM2 接收结果如图 14-6 所示。

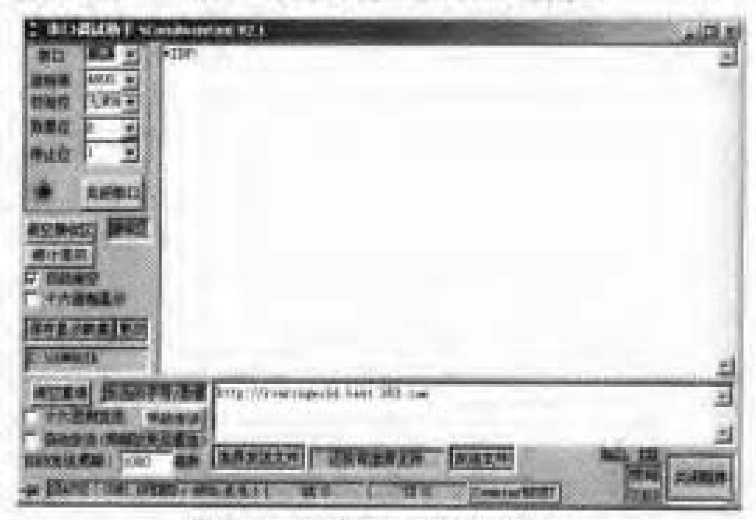


图 14-6 通过 COM2 接收的结果

14.3.2 通信步骤及相关函数介绍

从上面的例子我们可以简单地看出，一个完整的串口通信基本步骤如下：

- **产生一个串口对象**：用串口产生函数为串口产生一个串口对象。

在串口对象产生期间也可以配置其属性，如波特率、数据比特位数等。

- **连接外部设备**：用 fopen 函数连接串口对象和外部设备。

串口对象连接后，也可以通过设置属性、读数据及写数据来改变设备属性。

- **配置串口属性**：通过为串口属性分配值来得到串口对象属性值。

可以用设置函数来为属性赋值。实际上，我们能在串口对象产生期间或者产生后的任何时间配置属性值。如果接受默认值，则可以跳过这一步。

- **读写数据**：用读写函数我们能读写数据到设备中。

根据配置的属性值，用 fprintf 或 fwrite 来写数据到外设，用 fgetl、fgets、fread、fscanf 或 readasync 函数来从外设读取数据。

- **断开连接和清除**：当我们不再需要串口对象时，应该断开其和设备的关连，从内存移除，并从 MATLAB 工作区中清除掉。

下面就以上几个步骤分别加以详细介绍：

1．产生一个串口对象

用 serial 函数为串口产生一个串口对象。该函数的调用格式为：

- obj = serial('port')　产生一个与指定串口相关连的串口对象，如果串口不存在，或正在使用，关连将失败。
- obj = serial('port','PropertyName',PropertyValue,...)　产生一个限定属性名和属性值的串口对象，如果限定了无效的属性名和属性值，串口对象将不能产生，返回错误。

其中：参数'port'为串口名；'PropertyName'为串口属性名；PropertyValueA 为 PropertyName 所支持的属性值；obj 为串口对象。

当产生一个串口对象时，自动配置下面的属性值：

① Type 属性。

② Name 属性，即串口通信函数中的名字。

③ Port 属性，即串口名字。

【例 14-1】 用下面的例子产生串口对象 s1，并且与串口 COM1 相关连。

```
s1 = serial('COM1')     % 类型、名字和串口属性被自动配置
get(s1,{'Type','Name','Port'})
```

函数返回：

```
ans =
     'serial'     'Serial-COM1'     'COM1'
s2 = serial('COM2','BaudRate',1200,'DataBits',7);     % 产生对象时限定属性
```

2．连接外部设备

连接串口对象和外部设备

在用串口对象读写数据前，必须先通过函数 fopen 将串口对象和设备相连。连接以后，一些属性值是只读的，如 InputBufferSize 及 OutputBufferSize 等，所以必须在使用 fopen 函数前设置好。我们可以用下面的语句来测试或验证连接状态。

```
s.status
ans =
    open
```

一旦串口对象和设备连接成功，我们就可以读写数据，如图 14-7 所示。

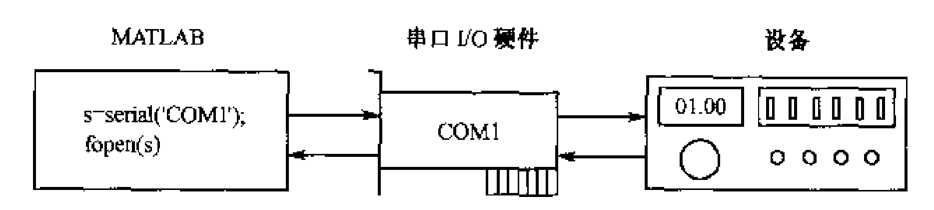


图 14-7　MATLAB 与设备之间读写数据

fopen 函数的调用格式为：

fopen(obj)

其中 obj 为串口对象或串口对象向量。

【例 14-2】 本例产生串口对象 s，用 fopen 连接 s 和设备，读写文本数据，然后断开 s 和设备的连接。

程序如下：

```
s = serial('COM1');
fopen(s)
s.Status
fprintf(s,'*IDN?')
idn = fscanf(s);
fclose(s)
```

函数返回状态为：

```
ans =
        open
```

对象连接后，我们能通过配置属性值、读数据和写数据来改变设备的一些设置。

3. 配置串口属性

通过为串口属性分配值来建立希望的串口对象属性值。使用 set 函数配置串口属性，该函数的调用格式为：

- set(obj)：函数显示 obj 的所有可配置属性值。
- props = set(obj)：返回 obj 的所有可配置属性值到 props 结构中。
- set(obj,'PropertyName')：显示 PropertyName 的有效值。
- props = set(obj,'PropertyName')：返回 PropertyName 的有效值到 props 结构。
- set(obj,'PropertyName',PropertyValue,...)：配置多个属性值。

一旦串口对象建立，就可以用 set 函数显示所有的可配置属性值。

```
s = serial('COM1');
set(s)
```

显示如下：

```
ByteOrder: [ {littleEndian} | bigEndian ]
BytesAvailableFcn: string -or- function handle -or- cell array
BytesAvailableFcnCount
```

BytesAvailableFcnMode: [{terminator} | byte]
ErrorFcn: string -or- function handle -or- cell array
InputBufferSize
Name
ObjectVisibility: [{on} | off]
OutputBufferSize
OutputEmptyFcn: string -or- function handle -or- cell array
RecordDetail: [{compact} | verbose]
RecordMode: [{overwrite} | append | index]
RecordName
Tag
Timeout
TimerFcn: string -or- function handle -or- cell array
TimerPeriod
UserData

SERIAL *specific properties:*
BaudRate
BreakInterruptFcn: string -or- function handle -or- cell array
DataBits
DataTerminalReady: [{on} | off]
FlowControl: [{none} | hardware | software]
Parity: [{none} | odd | even | mark | space]
PinStatusFcn: string -or- function handle -or- cell array
Port
ReadAsyncMode: [{continuous} | manual]
RequestToSend: [{on} | off]
StopBits
Terminator

【例 14-3】 下面的例子显示了一些配置串口对象 s 属性的方式。

```
s = serial('COM1');
set(s,'BaudRate',9600,'Parity','even');  %设置波特率，奇偶校验。
set(s,{'StopBits','RecordName'},{2,'sydney.txt'});  %设置停止位。
set(s,'Parity')
[ {none} | odd | even | mark | space ]
```

实际上，我们能在对象产生时或产生后的任何时候配置其中一些属性。我们也可以根据需要采用缺省值而跳过此步骤。

4．读写数据

用下面的读写函数能将数据写入到设备中或从设备读取数据。

（1）fprintf 函数：写文本数据到设备。

- fprintf(obj,'cmd')：写字符串 cmd 到与 obj 相连的设备中。缺省的格式为%s\n；写操作是同步的，即阻塞命令行直到执行完成。
- fprintf(obj,'format','cmd')：将由 format 格式限定的字符串 cmd 写到与 obj 相连的设备中。Format 是 C 语言的转义字符。
- fprintf(obj,'cmd','mode')：写字符，'mode'限定命令行模式。如果 mode 是 sync，cmd 被同步写，命令行被阻塞；如果 mode 是 async，cmd 被异步写，命令行不被阻塞；mode 缺省为

同步。

- fprintf(obj,'format','cmd','mode')：写由format格式限定的字符串，如果mode是sync，cmd被同步写，如果mode是async，cmd被异步写。

（2）fscanf函数：以文本格式从设备读数据。

- A = fscanf(obj)：从与obj相关联的设备读数据，返回到A。数据被转换为有%c格式的文本。
- A = fscanf(obj,'format')：读数据，并根据format转换。
- A = fscanf(obj,'format',size)：读数据，大小由size限定。
- [A,count] = fscanf(...)：返回读的数量到count中。
- [A,count,msg] = fscanf(...)：如果读操作没有完全成功，返回警告信息到msg。

【例14-4】 产生串口对象s，连接到外部设备，用fprintf函数写有格式的字符串：fprintf function write format text to device。

程序为：

```
s = serial('COM1');
fopen(s)
fprintf(s,' fprintf function write format text to device ')
out = fscanf(s)
```

设备接收到的数据显示如图14-8所示：



图14-8　设备接收到的数据

返回值为：

```
out =
     fprintf function write format text to device
```

（3）fwrite函数：写二进制数据到设备。

- fwrite(obj,A)：写二进制数据A到与obj相关连的设备中。
- fwrite(obj,A,'precision')：写二进制数据A，precision限定了精度，精度控制被写值的比特数及其类型，如integer、floating-point或character。如果precision没有限定，则为uchar。
- fwrite(obj,A,'mode')：写二进制数据，同fprintf一样，mode限定了同步或异步的方式。
- fwrite(obj,A,'precision','mode')：写带有精度和mode限定的二进制数据。

（4）fread 函数：以二进制格式从设备读数据。

- A = fread(obj,size)
- A = fread(obj,size,'precision')
- [A,count] = fread(...)
- [A,count,msg] = fread(...)

函数各项意义同 fscanf。

另外还有读一行文本的函数 fgetl 和 fgets。两者的区别在于 fgetl 不处理结束符，而 fgets 要处理结束符。也可以用 readasync 函数异步读数。格式为：

```
readasync(obj)
readasync(obj,size)
```

【例 14-5】 产生串口对象 s，连接到外部设备，用 fread 函数读外设发送的二进制数据：ABC。

程序为：

```
s = serial('COM1');
fopen(s)
out = fread(s,3,'uint8')
```

函数返回如下：

```
out =
    97
    98
    99
```

即为 ABC 的 ASCII 码。

5．断开连接和清除

当我们不再需要串口对象时，应该断开其和设备的关连，从内存移除，并从 MATLAB 工作区中清除掉，可使用下面 3 个函数。

（1）fclose 函数断开串口对象与设备的关连。其调用格式为：

```
fclose (serial)
```

obj 为一串口对象或串口对象向量。此时串口对象仍然存在。

（2）delete 函数从内存中移除串口对象。调用格式为：

```
delete (serial)
```

obj 为一串口对象或串口对象向量。

（3）clear 函数从 MATLAB 工作区中清除串口对象。调用格式为：

```
clear (serial)
```

14.4 应用实例

通过上面的介绍，我们对相关步骤和函数有了详尽的了解，下面将通过用 MATLAB 串口 I/O 函数编写收和发的工程实例，来全面深入地了解其用法。

14.4.1 实例 1——与示波器通信

【例 14-6】 本例将示范怎样和串口仪器通信，并读写文本数据。仪器为双通道的 Tektronix

TDS 210 示波器，和 COM1 连接，一正弦信号输入到通道 2，现在我们的目的是度量输入信号的峰峰值。

（1）产生串口对象

```
s = serial('COM1')
    Serial Port Object : Serial-COM1
    Communication Settings
        Port:               COM1
        BaudRate:           9600
        Terminator:         'LF'
    Communication State
        Status:             closed
        RecordStatus:       off
    Read/Write State
        TransferStatus:     idle
        BytesAvailable:     0
        ValuesReceived:     0
        ValuesSent:         0
```

（2）连接到示波器

```
fopen(s)
```

（3）写读数据

用 fprintf 写*IDN?命令到示波器，然后用 fscanf 命令读回结果。

```
fprintf(s,'*IDN?')
idn = fscanf(s)
idn =
    TEKTRONIX,TDS 210,0,CF:91.1CT FV:v1.16 TDS2CM:CMV:v1.04
```

现在测试测量通道是哪一个，可能是通道 1 或者通道 2：

```
fprintf(s,'MEASUREMENT:IMMED:SOURCE?')
source = fscanf(s)
source =
        CH1
```

返回值表示示波器配置为从通道 1 返回测量数据，而输入信号连接到通道 2，所以必须配置示波器从通道 2 返回数据：

```
fprintf(s,'MEASUREMENT:IMMED:SOURCE CH2')
fprintf(s,'MEASUREMENT:IMMED:SOURCE?')
source = fscanf(s)
source =
        CH2
```

现在配置示波器返回峰峰值，然后请求测量值：

```
fprintf(s,'MEASUREMENT:MEAS1:TYPE PK2PK')
fprintf(s,'MEASUREMENT:MEAS1:VALUE?')
ptop = fscanf(s,'%g')
ptop =
    2.0199999809E0
```

（4）断开连接和清除

```
fclose(s)
delete(s)
clear s
```

14.4.2 实例 2——拆分输入数据

【例 14-7】 本例说明怎样用 strread 函数来拆分和格式化从设备读取的数据。仪器为双通道的 Tektronix TDS 210 示波器，和 COM1 连接。

（1）产生串口对象

```
s = serial('COM1')
   Serial Port Object : Serial-COM1
   Communication Settings
      Port:              COM1
      BaudRate:           9600
      Terminator:        'LF'
   Communication State
      Status:            closed
      RecordStatus:      off
   Read/Write State
      TransferStatus:    idle
      BytesAvailable:    0
      ValuesReceived:    0
      ValuesSent:        0
```

（2）连接到示波器

```
fopen(s)
```

（3）写读数据

用 fprintf 写 RS232?命令到示波器，然后用 fscanf 命令读回结果。RS232?请求返回 RS-232 设置，包括波特率、软件流控制、硬件流控制、校验类型及结束符。

```
fprintf(s,'RS232?')
data = fscanf(s)
data =
9600;0;0;NONE;LF
```

用 strread 函数来拆分并格式化数据变量为 5 个部分。

```
[br,sfc,hfc,par,tm] = strread(data,'%d%d%d%s%s','delimiter',';')
br =
9600
sfc =
0
hfc =
0
par =
'NONE'
tm =
'LF'
```

（4）断开连接和清除

```
fclose(s)
delete(s)
clear s
```

14.4.3 实例 3——计算机与计算机通信

【例 14-8】 本例假定外设为另外一个计算机，通过两个串口来模拟与外设的通信行为，为了更好的理解串口 I/O，发送程序采用 MATLAB 编写，用来模拟向外设发送数据文件；接收程序用“串口调试助手”，用来 MATLAB 发来的数据，将其保存为数据文件，并用 MATLAB 将数据文件作图（假定数据文件为一简单正弦函数图形文件）。

（1）先产生正弦函数数据文件，保存为 sin.bin。

程序如下：

```
x=[0:0.01:2*pi];
y=sin(x);
fid=fopen('f:/sin.bin','w');        %在 f 盘上产生文件 sin.bin
count=fwrite(fid,x,'float32');      %写变量 x 到文件
count=fwrite(fid,y,'float32');      %写 y 值到文件
fclose(fid);                        %关闭文件
```

（2）编写发送数据函数，将数据文件发给外部设备。

程序如下：

```
%read data from file
fid=fopen('g:/sin.bin','r');                  %打开数据文件
[x,countx]=fread(fid,629,'float32');          %读数据到 x 中
[y,county]=fread(fid,629,'float32');          %读数据到 y 中
fclose(fid);                                  %关闭文件
%串口 I/O
s = serial('COM1');
set(s,'BaudRate',9600);                       %串口设置
set(s,'OutputBufferSize',4000)
fopen(s);
fwrite(s,x,'float32','async')                 %开始发送数据 x
fwrite(s,y,'float32','async')                 %开始发送数据 y
fclose(s)
delete(s)
clear s
```

接收如图 14-9 所示。

（3）将接收的数据保存为文件，用 MATLAB 编程读取数据作图如图 14-10 所示。

将上面的程序加上握手协议，就可以加以修改在实际通信中运用。当然，接收程序也可以用 MATLAB 编写。

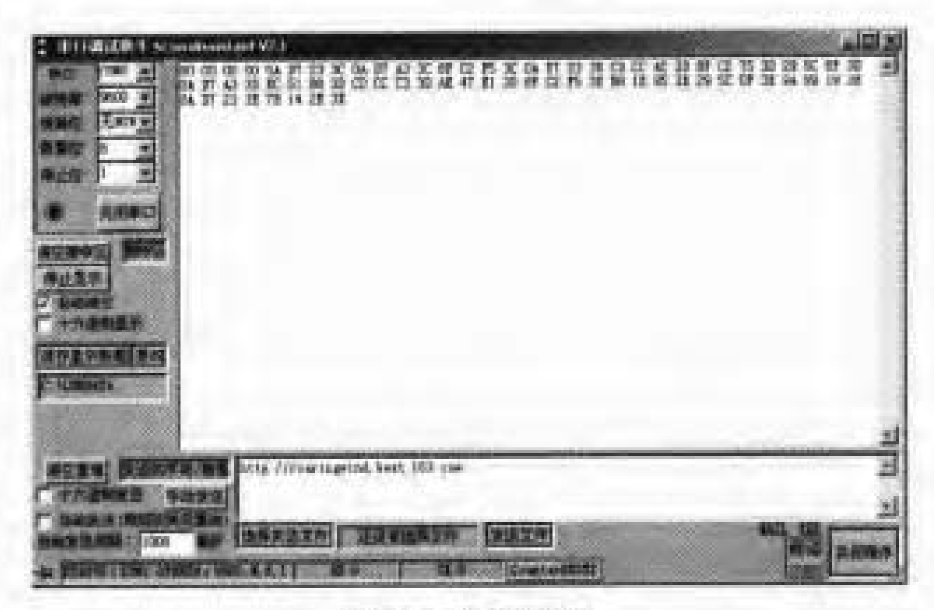


图 14-9 接收的结果

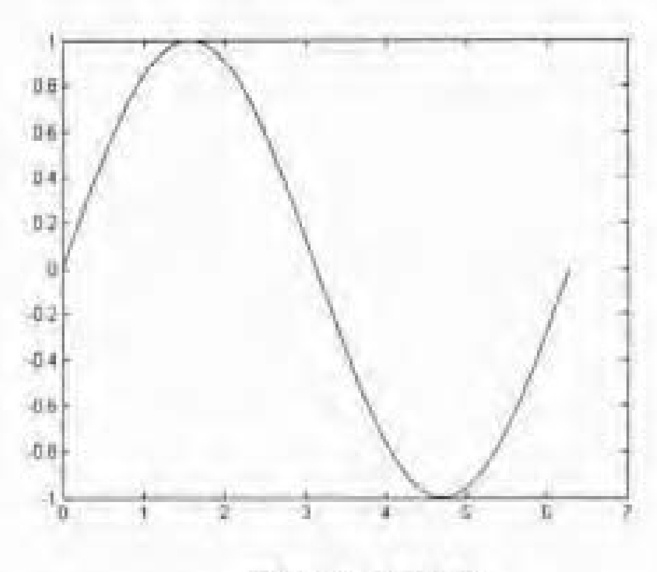


图 14-10 生成图形

14.5 串口 I/O 相关函数表

为了便于参考，将串口通信几个步骤中常用的 I/O 函数及其功能列于表 14-2。

表 14-2 常用串口 I/O 函数表

函 数	功 能
clear (serial)	从 MATLAB 工作区中清除串口对象
delete (serial)	从内存中移除串口对象
disp (serial)	显示串口对象总体信息
fclose (serial)	断开串口对象与设备的关连
fgetl (serial)	从设备读数据，忽略结束符
fgets (serial)	从设备读数据，包括结束符
fopen (serial)	将串口对象和设备关连

续表

函　数	功　能
fprintf (serial)	向设备写文本数据
fread (serial)	从设备读取二进制数据
fscanf (serial)	从设备读取数据，格式为文本
fwrite (serial)	向设备写二进制数据
get (serial)	返回串口对象的属性
instrcallback	当事件发生时，显示事件信息
instrfind	从内存中返回串口对象到工作区
isvalid	判断串口对象是否有效
length (serial)	串口对象向量的长度
load (serial)	装载串口对象及变量到工作区
readasync	异步从设备读数据
record	记录数据和事件信息到文件
save (serial)	保存串口对象和变量到 MAT 文件
serial	产生串口对象
serialbreak	向设备发送中断指示
set (serial)	设置或显示串口对象属性
size (serial)	串口对象矩阵的尺寸
stopasync	停止异步的读写操作

第 15 章　界面设计技巧

界面设计是 MATLAB 的主要难题之一，本章主要介绍一些这方面的实用技巧，包括使用外部控件、控件交互操作和将 MATLAB 图形显示到 VB 界面等。相信本章的学习会带给你一些美妙的感觉。

15.1　使用外部控件

本书第 6 章和第 8 章都已经介绍过在 MATLAB 中使用外部 ActiveX 控件的问题，并且第 8 章给出了一个在 VB 中创建 ActiveX 控件，然后在 MATLAB 中使用该控件的完整实例。使用外部 ActiveX 控件，可以无限扩展 MATLAB 界面编程的可能性。

第 6 章和第 8 章讨论的是如何用 actxcontrol 函数创建 ActiveX 控件。其实 MATLAB 7.0 在 GUIDE 中提供了一个 ActiveX 控件按钮，可以利用该按钮在设计区创建和绘制 ActiveX 控件。

下面的实例使用 GUIDE 中的 ActiveX 控件按钮创建微软的 MSFlexGrid 控件并利用它显示数据。

注意：使用 MSFlexGrid 控件，需要首先安装 Visual Basic 6.0。

首先打开 GUIDE，创建一个空白的界面，保存为 FlexGrid，在左侧的工具箱中单击 ActiveX 按钮以后，在设计区进行拖拉。此时会弹出 “Select an ActiveX Control” 对话框，如图 15-1 所示。

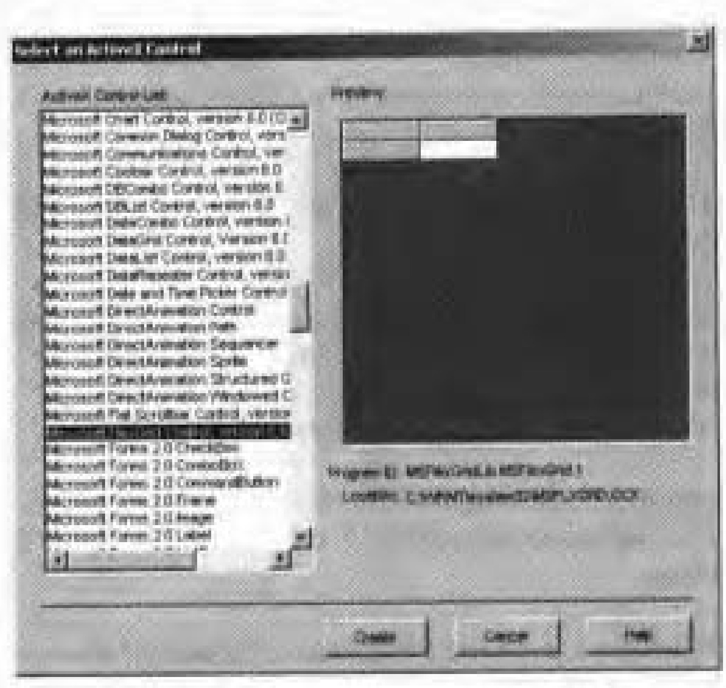


图 15-1　“Select an ActiveX Control” 对话框

对话框中的 “ActiveX Control List” 列表框中列出了当前在计算机上注册的所有 ActiveX

控件。在其中选择"Microsoft FlexGrid Control,version 6.0"选项，右侧显示该控件的外观预览。单击"Create"按钮，创建该控件。此时会在设计区按照先前拖拉操作时绘制的矩形的大小显示控件，但控件外观不显示出来，只显示有控件名称的空白区域，绘制了矩形的对角线。

继续在设计区添加静态文本 lblSize，其 String 属性值为"魔方矩阵的大小"；添加可编辑文本框 txtSize，其 String 属性值为 4；添加命令按钮 cmdShow，其 String 属性值为"显示魔方矩阵"。完成设计以后的程序界面如图 15-2 所示。

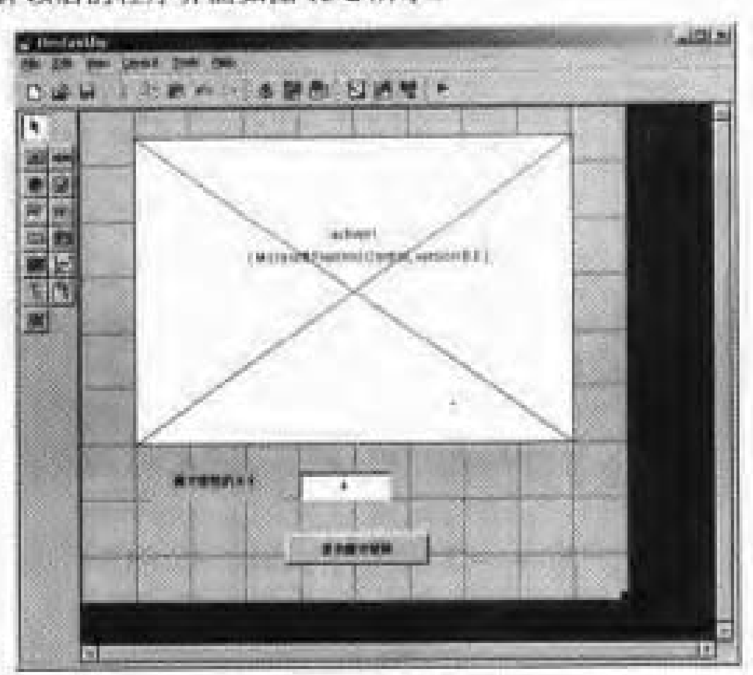


图 15-2 设计界面

然后修改 M 文件，添加各控件的回调代码，如下所示：

```
function varargout = FlexGrid(varargin)
%FLEXGRID M-file for FlexGrid.fig
%       FLEXGRID, by itself, creates a new FLEXGRID or raises the existing
%       singleton*.
%
%       H = FLEXGRID returns the handle to a new FLEXGRID or the handle to
%       the existing singleton*.
%
%       FLEXGRID('Property','Value',...) creates a new FLEXGRID using the
%       given property value pairs. Unrecognized properties are passed via
%       varargin to FlexGrid_OpeningFcn.   This calling syntax produces a
%       warning when there is an existing singleton*.
%
%       FLEXGRID('CALLBACK') and FLEXGRID('CALLBACK',hObject,...) call the
%       local function named CALLBACK in FLEXGRID.M with the given input
%       arguments.
%
%       *See GUI Options on GUIDE's Tools menu.   Choose "GUI allows only one
%       instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
```

```
% Edit the above text to modify the response to help FlexGrid
% Last Modified by GUIDE v2.5 25-Apr-2005 16:30:20
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',          mfilename, ...
                   'gui_Singleton',    gui_Singleton, ...
                   'gui_OpeningFcn', @FlexGrid_OpeningFcn, ...
                   'gui_OutputFcn',   @FlexGrid_OutputFcn, ...
                   'gui_LayoutFcn',   [], ...
                   'gui_Callback',    []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before FlexGrid is made visible.
function FlexGrid_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%              command line (see VARARGIN)

handles.mat_size=4;
handles.matrix=magic(4);

% Choose default command line output for FlexGrid
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes FlexGrid wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = FlexGrid_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in btnShow.
function btnShow_Callback(hObject, eventdata, handles)
% hObject    handle to btnShow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
num=handles.mat_size+1;
data=handles.matrix;
h=handles.activex5;
h.Cols=num;
h.Rows=num;
for m=1:num-1
    for n=1:num-1
        h.Row=m;
        h.Col=n;
        h.Text=num2str(data(m,n));
    end
end


function txtSize_Callback(hObject, eventdata, handles)
% hObject    handle to txtSize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txtSize as text
%        str2double(get(hObject,'String')) returns contents of txtSize as a double
handles.mat_size=str2double(get(handles.txtSize,'String'));
handles.matrix=magic(handles.mat_size);

guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function txtSize_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txtSize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

运行程序，在可编辑文本框中输入 5，单击“命令”按钮，会在 MSFlexGrid 控件中显示大小为 5 的魔方矩阵，如图 15-3 所示。

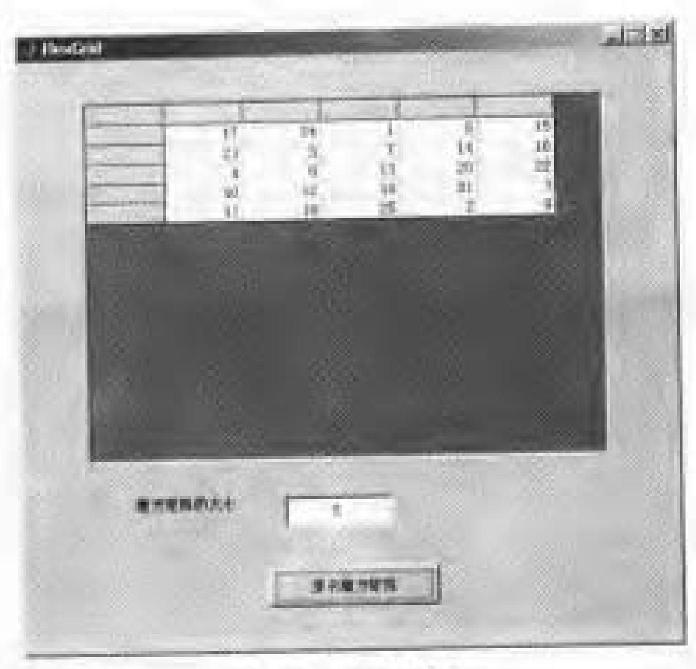


图 15-3 运行程序

15.2 控件的选择、移动、缩放和复制

利用 selectmoveresize 函数可以实现坐标系对象和控件图形对象的选择、移动、缩放和复制。实际编程时，将该函数作为按钮按下时的回调函数。

如下例所示，首先生成 peaks 数据的曲面图，然后将当前坐标系的 ButtonDownFcn 属性值设置为 selectmoveresize。

```
surf(peaks)
set(gca,'ButtonDownFcn','selectmoveresize')
```

生成 peaks 数据的曲面图。用鼠标单击坐标系，在坐标系周围显示 6 个黑色手柄，如图 15-4 所示。将鼠标光标放在手柄上，然后拖拉鼠标，可以缩小或放大坐标系和图形，如图 15-5 所示；将鼠标光标放在坐标系内图形以外的区域，然后拖拉鼠标，可以移动坐标系和图形。

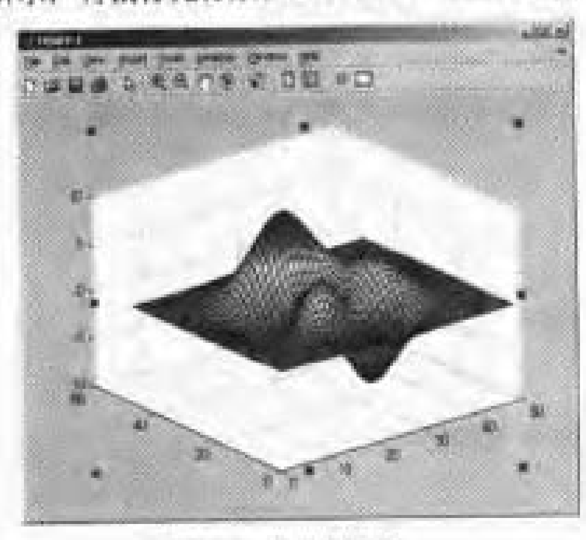


图 15-4 选择坐标系

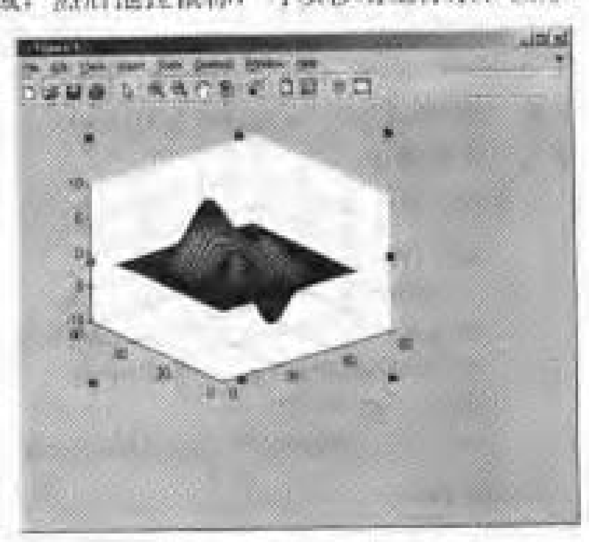


图 15-5 缩小坐标系和图形

对于控件，也可以进行类似的操作。例如

```
h=uicontrol
set(h,'ButtonDownFcn','selectmoveresize')
```

生成一个命令按钮，如图 15-6 所示。用鼠标右键单击按钮，然后按住右键的同时拖动鼠标，可以复制该按钮。如图 15-7 中复制了 3 个按钮。

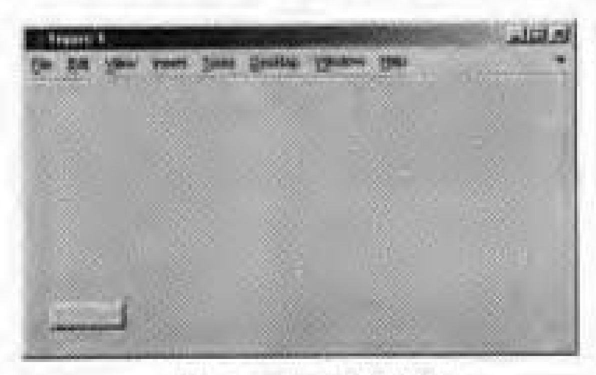


图 15-6　生成命令按钮

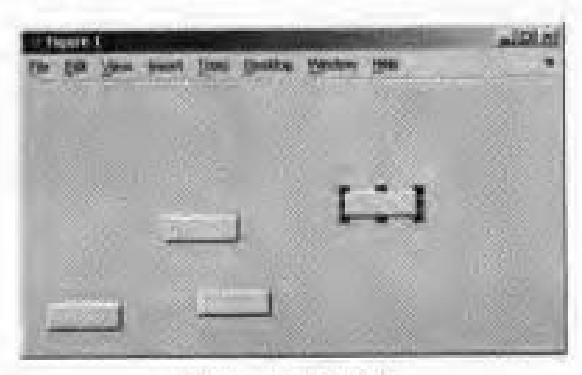


图 15-7　复制按钮

下面的语法格式：

```
A=selectmoveresize
```

返回一个包含下面两个字段的结构数组：

- A.Type　为一包含操作类型的字符串，可以是 Select, Move, Resize 或 Copy。
- A.Handles　为一列选中的句柄，或者对于 Copy 操作类型，为 $m\times2$ 的矩阵。该矩阵将原始句柄放在第 1 列，将新句柄放在第 2 列。

15.3　控件标题文本的换行

利用 textwrap 函数可以实现控件标题文本的换行。该函数返回给定控件的换行字符串矩阵，调用格式为：

- outstring=textwrap(h,instring)　返回一个换行了的字符串单元数组 outstring，它适应句柄为 h 的控件的内部大小。instring 是一个单元数组，每个单元包含一行文本。outstring 是一个单元数组格式的换行字符串矩阵。输入字符串的每个单元作为一幅图形进行处理。
- [outstring,position]=textwrap(h,instring)　返回控件的建议位置。position 为 x 和 y 方向上多行文本的宽度。

下面的例子将换行文本字符串放在控件中：

```
pos = [10 10 100 10];
h = uicontrol('Style','Text','Position',pos);
string = {'This is a string for the uicontrol.','It should be correctly wrapped inside.'};
[outstring,newpos] = textwrap(h,string);
pos(4) = newpos(4);
set(h,'String',outstring,'Position',[pos(1),pos(2),pos(3)+10,pos(4)])
```

生成图 15-8。

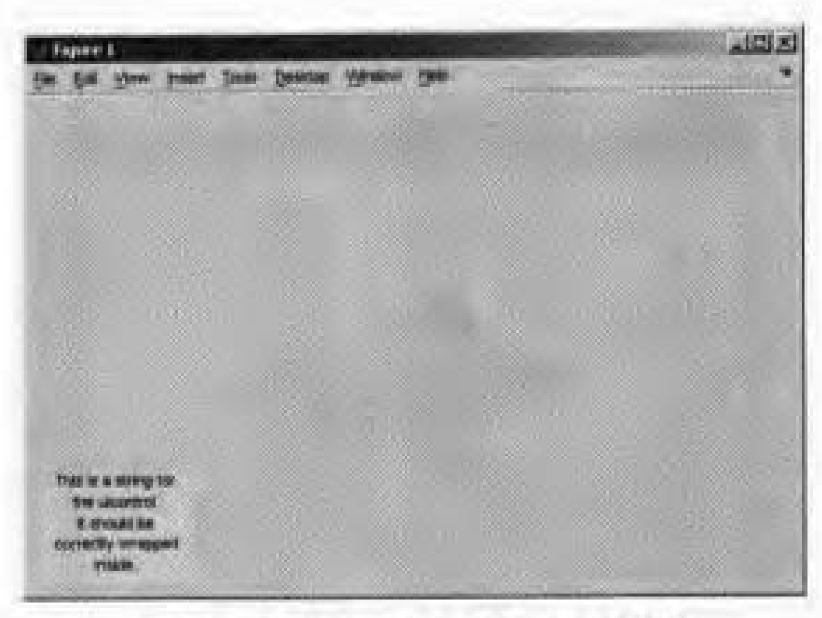


图 15-8　将换行文本字符串放到静态文本标签中

15.4　将 MATLAB 绘制的图形显示到 VB 界面上

本节介绍使用文件外部传输的方法将 MATLAB 绘制的图形显示到 VB 界面上。我们结合一个实例进行介绍。

在 VB 中创建一个标准 EXE 工程 Draw，窗体名称为 frmDraw。在窗体上添加一个图片框 imgDraw，其 Stretch 属性值为 True。添加一个命令按钮 cmdDraw，其 Caption 属性为"将 MATLAB 图形显示在 VB 界面上"。

在代码窗口添加下面的代码。

```
Private Sub cmdDraw_Click()
    Dim objMATLAB As Object
    Set objMATLAB = CreateObject("matlab.application")
    Dim strEnter As String
    Dim strCommand As String
    strEnter = Chr(13) & Chr(10)
    strCommand = "[x,y,z]=peaks(30);"
    strCommand = strCommand & strEnter
    strCommand = strCommand & "surfc(x,y,z)"
    strCommand = strCommand & strEnter
    strCommand = strCommand & "shading interp"
    strCommand = strCommand & strEnter
    strCommand = strCommand & "saveas(gcf,'C:\temp.bmp')"
    objMATLAB.execute (strCommand)
    imgDraw.Picture = LoadPicture("C:\temp.bmp")
    imgDraw.Refresh
    Set objMATLAB = Nothing
End Sub
```

代码首先创建 MATLAB 对象的实例及 MATLAB 命令字符串，然后用 execute 函数执行 MATLAB 命令，生成 peaks 数据的表面图和等值线图，并将它们保存在 C 盘下的 temp.bmp

文件中。最后在 VB 图片框中载入该图形，注销 MATLAB 对象。

运行程序，单击命令按钮，在图片框中显示生成的图形，如图 15-9 所示。

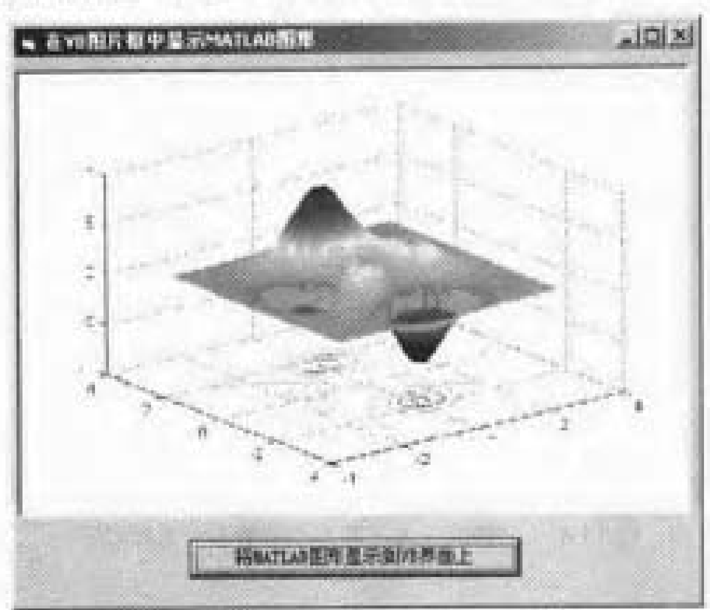


图 15-9　在 VB 界面上显示 MATLAB 图形